Oracle for UNIX Performance Tuning Tips

ORACLE7 for UNIX

Performance Tuning Tips

ORACLE for UNIX

Performance Tuning Tips

Part No. A22535-2

February 1993
First Revision September, 1994
Second Revision September, 1995

Authors: Chidori Boeheim, Tony Duarte, Mark Johnson

Contributors:

Andrew Holdsworth, Cary Millsap, John Frazzini, Cecilia Gervasio, Anna Leyderman, Angela Amor, Hamid Bahadori

Contents

I. Preface

Purpose

The purpose of this guide is to help users improve the performance of Oracle applications on UNIX systems. Oracle Corporation believes that following the tuning tips in this guide will result in a reasonably well-tuned Oracle environment but does not intend this guide as the definitive authority on tuning Oracle on UNIX.

If after reading this paper you wish to explore further, please see the Bibliography for more specialized documents. This paper includes material from the references as needed. Specialized help is also available from the Oracle Consulting Group; for more information contact your regional Oracle Consulting manager.

This revision features many new tips. New material has been added to cover the 7.1 parallel query and Parallel Server options. The Bibliography has been enhanced and annotated. Experience from Oracle's internal tuning and benchmarking groups as well as feedback from the UNIX Level 3 Performance Tuning Course has been added.

Audience

This guide is written for new as well as experienced Oracle Database Administrators (DBA's) and UNIX System Managers who are familiar with either Oracle or UNIX, but not necessarily both. New and part-time DBAs can use this guide to advance their tuning skills. Experienced, full-time UNIX System Managers may find this guide useful for validating tuning procedures they already follow and for discovering a few new ideas.

Using this Guide

This guide illustrates how to use features within Oracle products and UNIX to tune your Oracle database. More detailed descriptions may be found in the *Oracle7 Server Administrator's Guide*, the platform-specific *Oracle for UNIX Installation & Configuration Guides*, the *Oracle7 Server Tuning Guide, Release 7.2* and other references described in the Bibliography. Included here you will find basic instructions for using various performance monitoring tools and sample displays from them.

The guide clearly indicates whenever a procedure may optimize UNIX for Oracle at the expense of other applications. Responsibility for balancing the performance of Oracle and non-Oracle applications rests with the System Manager.

Before you begin the actual tuning, please read through this guide. If you are focusing on a particular tip, prepare the reports that may be needed from your system before tuning.

Feedback

The information in this guide has been taken from previously published technical documentation and from the personal experiences of Oracle/UNIX performance specialists. Oracle Corporation is very interested in your feedback, especially suggestions for tuning UNIX that will improve this guide. We are also interested in the problems, concerns, and ambitions of our customers.

Please complete and return the comment form in the back of this guide to provide us your feedback on the effectiveness of this guide and information about your system. If you have access to the Internet, you can send your comments electronically to Tony Duarte at address `aduarte@us.oracle.com`.

II. Steps for Tuning Oracle

The Oracle Server is a very sophisticated and highly tunable software product. You can meet your needs or maximize your results by tailoring the Oracle Server to your specific environment. Oracle applications often come with a long and varied history. Many Oracle for UNIX applications were developed on a system with a different configuration or ported from another platform. As they gain acceptance in the new environment they increase in scope and complexity. This growth may escape attention resulting in a poorly tuned Oracle system.

You can often improve performance dramatically by simply correcting common errors in an application or its configuration. We recommend a seven-step method for tuning Oracle applications as summarized below. You can find detailed information for each of the steps in the Bibliography. Measurements are an essential part of the tuning process.

Some of these steps require you to analyze Oracle Server performance statistics from *sqldba* (Monitor menu), *Server Manager*, or by using `utlbstat/utlestat` SQL scripts. Other suggestions refer to the X$tables and V$views that hold state information on a running Oracle instance. Additionally, some of the steps refer to tuning UNIX structures such as memory and disk I/O system. (*Server Manager* was used for most screen shots. On your system, *sqldba* or other monitoring software may be more convenient.) All of these sources of tuning information can change from release to release of the Oracle Server and underlying UNIX operating system. Often, the specific information in the tip is still present in slightly different form. Experiment a bit with the various measurement methods described in this document to learn how they work for your system.

Figure 1, Oracle Tuning Steps, graphically illustrates the Oracle tuning process. The height of each step indicates the relative gain expected from that step. While not to scale, this suggests that tips associated with application design and data access methods (Step 2 and Step 3) usually obtain the greatest performance gain.

The following steps list some of the major performance tuning tasks:

Step 1. Installing and Configuring Oracle

- Install and Configure Oracle using the Oracle Optimal Flexible Architecture (OFA) Rules

Step 2. Application Design

Step 3. Tuning Data Access

- Tune SQL Statements: TKPROF and EXPLAIN PLAN

Step 4. Tuning Memory Management

- Tune Number of Database Buffers and Redo Buffers
- Tune Data Dictionary Cache
- Reduce Swap-out and Page-out activities

Step 5. Tuning Disk I/O

- Distribute I/O and Applications
- Tune Number of Database Writers
- Check for Large Disk Request Queues
- Check for Disk and Tablespace Fragmentation

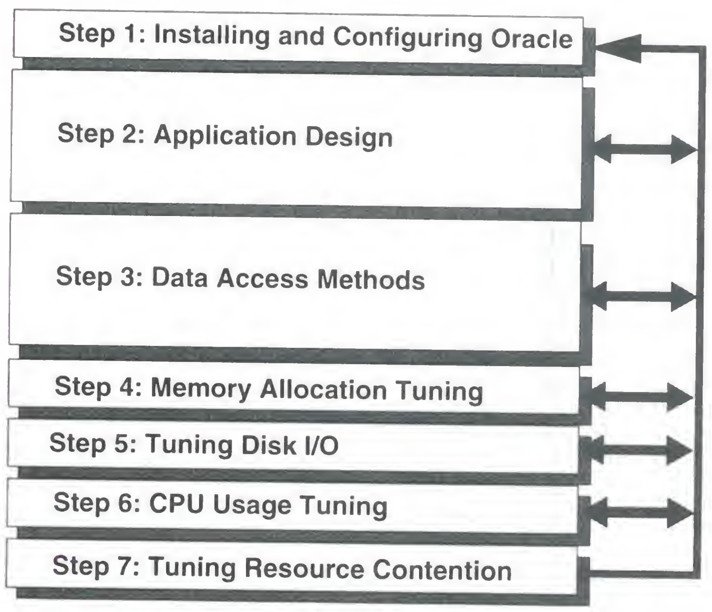

Step 1: Installing and Configuring Oracle
Step 2: Application Design
Step 3: Data Access Methods
Step 4: Memory Allocation Tuning
Step 5: Tuning Disk I/O
Step 6: CPU Usage Tuning
Step 7: Tuning Resource Contention

Figure 1: Oracle Tuning Steps

Step 6. Tuning CPU Usage

- Balance CPU loads
- Reorganize Usage Patterns

Step 7. Tuning Resource Contention

It is important to follow the steps in this order since decisions you make in one step may influence subsequent steps. For example, in Step 3 you may rewrite some of your SQL statements. These SQL statements may have significant bearing on parsing and caching issues addressed in Step 4. Also, disk I/O, which is tuned in Step 5, depends on the size of the buffer cache, which is tuned in Step 4. The components of the tuning process that have the greatest effects on other steps appear early in the method. For this reason, you should follow these steps in order.

Oracle Consultants often approach tuning from the opposite direction: they look at contention issues first. When arriving on site, they run a series of measurements to characterize the system and look for bottlenecks. This is because sometimes a single bottleneck is the primary cause of poor performance. Until it is detected and removed, other measurements will be badly distorted. Tip 53 and related tips discuss contention, while Tip 32, 33, and 43 show measurements to identify I/O and CPU bottlenecks. Once the system is free of a single, obvious bottleneck, tuning can be performed as described in this seven step process.

Since some performance gains made in later steps may pave the way for further improvements in earlier steps, additional passes through the tuning process or the iteration of some steps may be useful and sometimes necessary. Before beginning, you should decide what your tuning objectives are. Tuning can be an exciting, rewarding adventure. Remember to stop the iterative tuning effort once you have reached the objectives.

III. Identifying Performance Bottlenecks

It is important to understand where performance bottlenecks might occur before taking any steps to tune your Oracle system. Often, a system will be slow because a single component (the bottleneck) will limit overall performance. Three fundamental system resources can cause contention: memory (including cache), the I/O subsystem (disks and networks), and CPU (or CPU's for SMP's and MPP's). Figure 2, below, shows how these resources are interconnected in a typical system.

Memory contention can occur when processes require more memory than what is available on the system. To accommodate this shortage, the system performs paging and swapping (i.e., memory managing by moving processes or pages between memory and disk). It is important to monitor these activities since they

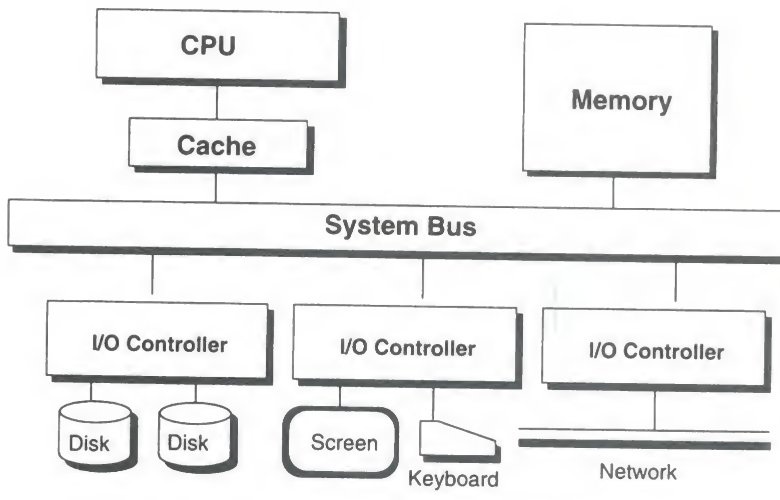


Figure 2: Computer System Components

can cause performance degradation. To detect any memory contention, you can monitor the memory activities of the Oracle Server using Sever Manager (see Tips 19, 20 and 21). To monitor paging and swapping activities, you can use *sar* on System V and *vmstat* on BSD UNIX (see Tips 12 and 13). To check swap space, you can use *pstat* on BSD UNIX and *swap* on SYSTEM V (see Tip 14).

Disk I/O contention may be the result of poor memory management (with subsequent paging and swapping), or poor distribution of tablespaces and files across disks. In a well tuned system, the I/O load is spread fairly evenly across all of the disks, with none much more heavily loaded than the others. On System V, you can use *sar* to get information about I/O activities such as request queues to disks while *iostat* can provide you with similar information on BSD UNIX (see Tips 32 and 33). You can also monitor I/O activities through the *sqldba* Monitor Menu function (see Tip 33) and `utlbstat/utlestat` (see Tip 36). Balancing Disk I/O can be done by striping across several disks (see Tips 27-30) or moving files to unloaded disk drives (see Tip 34). Network contention is not discussed in this guide.

The CPU is also one of the components of the computer system for which processes may contend. Although the UNIX kernel can allocate the CPU effectively most of the time, many processes compete for CPU cycles. If your system has more than one CPU (multiprocessor environment), there may be different levels of contention on various CPUs. One way to measure CPU loads is to use *sar* on System V and *vmstat* and *iostat* on BSD UNIX (see Tip 43).

Contention can also be caused for Oracle resources: Typically locks and latches. To identify those bottlenecks most quickly, you can use the Oracle v$ tables as described in Tip 53. You can also monitor the contention for Oracle latches through `utlbstat/utlestat` (see Tip 60).

Over time, both the database size and number of users tends to grow. What was once a lightly loaded system gradually (or not so gradually) slows down. Starting use of parallel query can dramatically increase the load on every component of a typical computer system. As a result, checking for bottlenecks must be done from time to time to ensure that your system stays balanced.

IV. PARALLEL Tuning Tips

The Oracle Parallel Server option allows users on multiple nodes to access a common database simultaneously. It enables multiple Oracle7 Server instances to run on different nodes of a UNIX Cluster while providing concurrent access to common data. The Oracle parallel query option splits up many kinds of queries (and some DDL like CREATE INDEX and CREATE TABLE AS SELECT) into parts that can be executed in parallel on all the processors available. The Oracle Parallel Sever and parallel query options provide the opportunity to gain performance by supporting more parallel hardware architectures and by enabling more complete use of the existing hardware parallelism than previous versions of the Oracle Server. This version of the document has been updated to include tips appropriate for the two parallel options. In general, performance tuning tips remain valid when using the parallel options. Tuning information for parallel query can be found in Tips 10, 45, and 59 and in the *Oracle 7 Server Tuning Guide* Chapter 8. Additional tuning information for the parallel server can be found in the *Oracle 7 Parallel Server Administrator's Guide*, Appendix D.

When tuning highly parallel systems, contention issues are very important. In particular, Tips 62 to 66 discuss contention reduction for the Oracle Parallel Server option. Parallel query will scale best if data is striped across many disks. Unless this is done (Tips 28 to 30), parallel query performance will probably be disappointing.

V. Step 1: Installing and Configuring Oracle

Carefully configuring the system and designing the database can reduce bottlenecks and performance problems. After consulting with hundreds of Oracle sites since 1990, the Oracle Consulting Core Technologies team developed a set of requirements and rules to guide database administrators in their system design. This section borrows heavily from their work.

Tip No. 1: Properly Size the System

If you are purchasing a system to run the Oracle Server, proper configuration is essential. The system should be balanced, with enough processors, memory and disks to meet your projected transaction rates. For example, in an effort to provide the lowest possible initial cost, it is common for system vendors to meet the disk capacity requirements with the fewest disk drives possible. This may limit your ability to use the parallelism in the server, because the disks can become a bottleneck (Tip 2).

It is also important to properly size and configure your UNIX operating system for Oracle. Unless you started with a preconfigured solution like Workgroup Server, you will normally have to rebuild the UNIX kernel for Oracle. Please see the *Oracle7 for UNIX Installation & Configuration Guide* for configuration information appropriate for your computer system.

Tip No. 2: Use I/O Rates to Calculate Disk Drives Needed

When determining how many disks are needed, calculate based on the throughput needed to sustain transaction rates, as well as the database size.

For OLTP applications, use the results from TKPROF reports (Figure 3) to estimate the number of physical I/Os required per transaction. For example, if four physical I/Os are needed on average for one transaction, and your application will need 100 transactions per second, you will need to be able to do about 400 disk I/Os per second. One disk can do about 50 random I/Os per second, so a system to run this example would require at least 8 (400/50) disk drives with the data files balanced across all of them.

You also want to spread disk drives across many controllers. Most disk controllers can only sustain two drives doing sequential reads at one time. If tables are to be used for parallel queries, no more than two disks should be active on a single controller.

Tip No. 3: Install an OFA-Compliant Oracle Database

The Oracle OFA (Optimal Flexible Architecture) rules provide a way to install Oracle for optimal system performance that doesn't diminish with growth. The OFA eliminates data dictionary free space fragmentation, isolates other fragmentation, and minimizes contention. The OFA also provides administrative advantages. Configuration tuning and database component manipulation (like backup) are simplified. Database growth is easier to predict and accommodate, and you can easily integrate new databases or hardware. The Oracle installer can be used to install an OFA compliant database by supplying appropriate responses to the prompts, followed by some fixups. OFA is described in Appendix 1.

Tip No. 4: Avoid Dynamic Space Management

The data in an Oracle database and information about the data is stored in storage units called *segments*. These segments are used for tables, indexes, rollback segments, and temporary segments. The basic unit of storage is a fixed sized Data Block containing *n* x (1024) bytes of information. Segments grow dynamically by allocating new extents, counted in units of Data Blocks (but specified in bytes) per extent. If too many extents are allocated, the information can become fragmented. Performance will suffer if there is excessive dynamic space management. Following the tips below will allocate your database to minimize the number of dynamically allocated extents, and substantially reduce any need for defragmentation. For more information on Oracle Server data structures, see the *Oracle7 Server Concepts Manual*, Chapters 3 and 4.

Preallocate enough space for tables, indexes, rollback segments, and temporary segments by using appropriate storage parameters such as INITIAL, NEXT, MINEXTENTS, MAXEXTENTS, and PCTINCREASE. To check the default parameters, use the SQL*DBA command:

```
SQLDBA> SELECT * FROM DBA_TABLESPACES;
```

Adjust PCTFREE to allocate space in the data blocks. You can raise PCTFREE to leave enough room for updates and inserts to avoid chaining blocks or lower PCTFREE to save space for read-only data.

Tip No. 5: Initialize Data and Index segments properly

The initial database installation should set INITIAL large enough for the initial data and index sizes plus the PCTFREE in each Data Block. Use NEXT=INITIAL (Oracle's benchmarking team often use a value about one-tenth the size of the file.). Choose INITIAL and NEXT values from the list 256K, 512K, 1M, 2M, 4M, 8M, 16M, 32M, etc. to minimize tablespace fragmentation. PCTINCREASE should normally be set to zero. The number of possible extents is determined by the block size. See the table in Tip 18 for information on the number of extents per block.

Tip No. 6: Initialize Temporary and Rollback segments properly

Temporary segments and rollback segments are used dynamically. They grow and shrink and are created and dropped frequently. Temporary segments should be configured with INITIAL set to $ks + b$, where k is an integer from 1 to 20, s is the sort area (in data blocks) and b is 1 data block. NEXT should be set to ks, and PCTINCREASE set to 0.

Rollback segments should be allocated initially as 20 to 40 equal sized extents (MINEXTENTS= 20 to 40). Each extent (INITIAL and NEXT) should be about 0.5% to 0.25% the size of the largest active table. MAXEXTENTS should be set to a high value. Try and allocate at least one rollback segment per available disk drive to spread the I/O as widely as possible. One rule of thumb is to create one rollback segment per four *active transactions*, but never create more segments than your instance's maximum number of concurrently *active transactions* and never create more than 50. Chapters 7, 8 and 9 of the *Oracle7 Server Administrator's Guide* give additional information on configuring data objects.

VI. Step 2: Application Design

Database design is the most important factor in determining the ultimate performance of your applications. There are numerous references available. *The Oracle7 Server Application Developer's Guide*, Ch. 1 offers advice and additional references. There is also a recent book (the bee book because of the cover illustration) from O'Reilly & Associates, Inc. called *Oracle Performance Tuning* by Peter Corrigan and Mark Gurry that covers application tuning and data access methods in detail.

Tip No. 7: Store LONG data separately

Place each LONG (and LONG RAW) column in a table separate from any other associated data. This can prevent SQL statements from scanning LONG's during full table scans. For the few rows that are returned, it is more efficient to do an indexed SELECT of only the rows that are needed.

Tip No. 8: Reuse SQL statements

Every SQL statement must be parsed and an execution plan chosen for it before it can be executed. The Oracle7 Server holds parsed SQL statements in a special part of shared memory called the library cache. These statements will be reused if the "identical" SQL statement (from any user) is processed "shortly" thereafter. Reusing a pre-parsed statement provides a substantial performance boost. "Identical" means exactly the same sequence of characters; case (upper or lower case), spaces, tabs, linefeeds, etc. included. Writing reusable SQL can substantially improve the performance of the library cache. Similar SQL statements can often be made identical using bind variables and assigning different values to the bind variable. For example:

```
SELECT :name FROM EMP WHERE EMP_NO = :variable;
```

replaces:

```
SELECT :name FROM EMP WHERE EMP_NO = 135;
```

and

```
SELECT :name FROM EMP WHERE EMP_NO = 137;
```

with one statement that may be found in the library cache.

When many users execute the same program, substantial reuse is almost certain to occur, especially if the library cache is large enough (Tip 23). Application development systems like Oracle Forms4 and Pro*C have "identical" SQL statements from user to user. Good software engineering practices such as using common libraries and sharing code will maximize the reuse of SQL statements.

Our tuning specialists have found that application code developed and tuned for versions 5 and 6 of the Oracle Server can frequently benefit significantly from retuning for SQL statement reuse.

VII. Step 3: Tuning Data Access

Before you start tuning system memory, disk I/O, and CPU, it is important to make sure that your application is well designed and well written. Tuning SQL statements is an important (thought not UNIX specific) component of a good design. The Bibliography lists several references for this step of the tuning process.

Tip No. 9: Look for Poor SQL Statements

1. Identify the SQL statements which need tuning.

   This step can be performed by turning on global database tracing. Set `SQL_TRACE = TRUE` and `TIMED_STATISTICS = TRUE` in the `initdb<n>.ora` file. When the database is restarted, trace information will be sent to files in the `USER_DUMP_DEST` directory. Be forewarned that a large volume of information will be generated and database operation will slow down during the trace. After running with trace enabled for enough time to characterize the operation of the database, tracing is turned off to restore database performance. (Leave `TIMED_STATISTICS = TRUE` in `initdb<n>.ora` for any timing and tracing work. However, leaving it in does add overhead to many low-level calls. A decision to turn off `TIMED_STATISTICS` must be made with care.)

   Statistics for a single session may be more meaningful, especially if you are focusing on a single application, and will certainly cut down on the amount of information gathered. To gather statistics for a single session, issue the SQL statement: `ALTER SESSION SET SQL_TRACE = TRUE.`

   The utility program `TKPROF` with the option sort=EXECPU can then be run on those trace files. This is often done with a command of the form shown below executed from the `USER_DUMP_DEST` directory.

```
$ COUNTER=0
$ for i in *
> do
> tkprof $i T$COUNTER sort=EXECPU
> COUNTER= 'expr $COUNTER + 1'
> done
```

   The output files can then be examined to characterize the type of database accesses. Typically, sites try to identify:

   - Time consuming operations. By examining frequency and duration of operations performed on the database, SQL that requires tuning can be identified.

```
=============================================================================
select nvl (cc.template_id, ''),
nvl (bal.currency_code, ''),
bal.set_of_books_id,
SEGMENT1, SEGMENT2, SEGMENT3, SEGMENT4,
nvl (begin_balance_cr,0), nvl(begin_balance_dr,0),
nvl (budget_version_id,''),
nvl (actual_flag,''),
nvl (period_name,''),
nvl (period_net_cr, 0), nvl(period_net_dr,0)
from GL_CODE_COMBINATIONS cc, GL_BALANCES bal
where bal.code_combination_id = cc.code _combination_id and
nvl (bal.translated_flag,'x') != 'R' and
bal.set_of_books_id in (1) and
(nvl (cc.SEGMENT1,'') between '2' and '2' ) and
(nvl (cc.SEGMENT2,'') between '60' and '60' ) and
(nvl (cc.SEGMENT3,'') between '3100' and 'ST31' ) and
(nvl (cc.SEGMENT4,'') between '0000' and 'S602' ) and
( (period_name='May-91' and actual_flag='A' )
or ( period_name='May-91' and budget_version_id=1003
        and actual_flag='B')
or ( period_name='May-90' and actual_flag='A' ) )

              count      cpu      elap      phys        cr      cur     rows
Parse:            0        0         0         0         0        0
Execute:          1        0         0         0         0        0        0
Fetch:          107     4393     80000     21330    353615        0     2139
=============================================================================
Execution plan:
CONCATENATION
NESTED LOOPS
TABLE ACCESS (BY ROWID) OF 'GL_BALANCES'
INDEX (RANGE SCAN) OF 'GL_BALANCES_N2' (NON-UNIQUE)
TABLE ACCESS (BY ROWID) OF 'GL_CODE_COMBINATIONS'
       INDEX (UNIQUE SCAN) OF 'GL_CODE_COMBINATIONS_U1' (UNIQUE)
NESTED LOOPS
TABLE ACCESS (BY ROWID) OF 'GL_BALANCES'
INDEX (RANGE SCAN) OF 'GL_BALANCES_N2' (NON-UNIQUE)
TABLE ACCESS (BY ROWID) OF 'GL_CODE_COMBINATIONS'
       INDEX (UNIQUE SCAN) OF 'GL_CODE_COMBINATIONS_U1' (UNIQUE)
NESTED LOOPS
TABLE ACCESS (BY ROWID) OF 'GL_BALANCES'
INDEX (RANGE SCAN) OF 'GL_BALANCES_N2' (NON-UNIQUE)
TABLE ACCESS (BY ROWID) OF 'GL_CODE_COMBINATIONS'
       INDEX (UNIQUE SCAN) OF 'GL_CODE_COMBINATIONS_U1' (UNIQUE)
```

Figure 3: Sample TKPROF/EXPLAIN PLAN Output

- Type of access: select, insert, update. This information is used to determine if features such as mirroring and indexing should be used. It can also suggest how data may be separated into tables.
- Commonly joined data. This information is used to determine if clustering, indexing, or data denormalization should be used.

2. Iteratively tune those statements.

One very common performance problem is caused by a SQL statement that consumes resources well beyond its reasonable share. To identify these statements, you can run a TKPROF report for Oracle application processing. SQL statements with excessively long execution times or a *Consistent Reads per Rows Returned* (*CR/Rows*) ratio greater than 15 warrant additional attention.

Figure 3 shows a sample TKPROF/EXPLAIN PLAN output from a poorly tuned SQL statement. In this example, *CPU* time for the fetch was about 44 seconds and the *CR/Rows* ratio was 165. Both values are excessive. Examining the data distribution and available indexes proved that the SQL optimizer chose an inefficient execution plan. Choosing a different index greatly improved the fetch times; *CPU* time dropped to 21 seconds and the *CR/Rows* ratio fell to about 82. Tuning SQL statements can often double the performance of the entire application.

EXPLAIN PLAN is very helpful for determining which index, if any, is used for specific queries. The proper use of indexes alone can often result in significant performance gain. For details on SQL trace facility, `TKPROF` and `EXPLAIN PLAN`, consult the *Oracle 7 Server Tuning Guide, Appendix A*.

The `utlbstat/utlestat` SQL scripts can be used to capture a snapshot of database performance statistics. This information is best gathered after an application is fully up and running. These SQL scripts can help identify database internal resource during application execution. If tracing is enabled during the `utlbstat/utlestat` interval, the statistics will include the tracing overhead. Use caution interpreting the results or run `utlbstat/utlestat` separately from SQL tracing. See Appendix 2 for more information on how to use `utlbstat/utlestat`.

```
count   = number of times OPI procedure was executed
cpu     = cpu time executing in hundredths of seconds
elap    = elapsed time executing in hundredths of secs
phys    = number of physical reads of buffers (from disk)
cr      = number of buffers gotten for consistent read
cur     = number of buffers gotten in current mode (usually for update)
rows    = number of rows processed by the OPI call
```

Figure 4: Key to TKPROF/EXPLAIN PLAN Output

Tip No. 10: Use the Oracle Parallel Query Option

For many queries, especially the ad hoc queries found in a decision support system, a full table scan is a necessary part of the query processing. On a Symmetrical Multiprocessor (SMP) or Massively Parallel Processor (MPP), parallel query can potentially apply most of the available CPU resources to a single query. This can improve performance by 3000% or more. If the processor consistently uses less than 40% *(usr + sys)* time (Tip 43), or if processors can be added to the system, the parallel query option may help performance. Tips 45 and 59 offer advice on setting up and tuning parallel query.

Because many processors are simultaneously working on data from a single table when processing a parallel query, it is vitally important that each table be distributed across as may disks as possible or disk access will become a bottleneck. Tips 32, 33 and 43 describe how to identify disk and cpu bottlenecks often associated with an untuned parallel query. Tips 28, 29, and 30, describe ways to distribute the data files across multiple disk drives.

Some super-linear query results have been reported. (Super-linear query results are results that speed up more than the number of processors. In one case, a customer observed a 5.6 times speedup on a 4 processor SMP.) The increased sort area available contributes to these results. Each query server has the same size sort area as an Oracle shadow processor. If the sort part of the query happens to go from mostly on disk to mostly in memory, a dramatic performance gain may result. See the *Oracle7 Tuning Guide, Chapter 13* for more information on tuning the sort area for parallel query.

Tip No. 11: Use Parallel Features Creatively

The Oracle7 Server parallel features can be used creatively to do more than speed up queries, for example, many Oracle RDBMS installations regularly delete data from the database. This happens, for example, during a nightly purging of outdated records. With the parallel options to Oracle, we have discovered that it can often be more effective to select all the other data in the database using the parallelized *create table as select* using the *unrecoverable* option, rather than deleting the unwanted records. This is faster because of the greater efficiency of the parallelization, as well as the suppression of redo information. Rebuilding the indexes can also be done in parallel. See the *Server Administrators Guide for Release 7.2* for additional information.

VIII. Step 4: Tuning Memory Management

Since memory access is much faster than disk access, it is desirable for data requests to be satisfied by access to memory rather than access to disk. For best performance it is advantageous to store as much data as possible in memory, rather than on disk. However, memory resources on your operating system are likely to be limited. Tuning memory allocation and management involve distributing available memory to Oracle memory structures. The memory tuning process often begins by tuning paging and swapping space (Tips 13-15) to determine how much memory is available, then tuning the Oracle data structures.

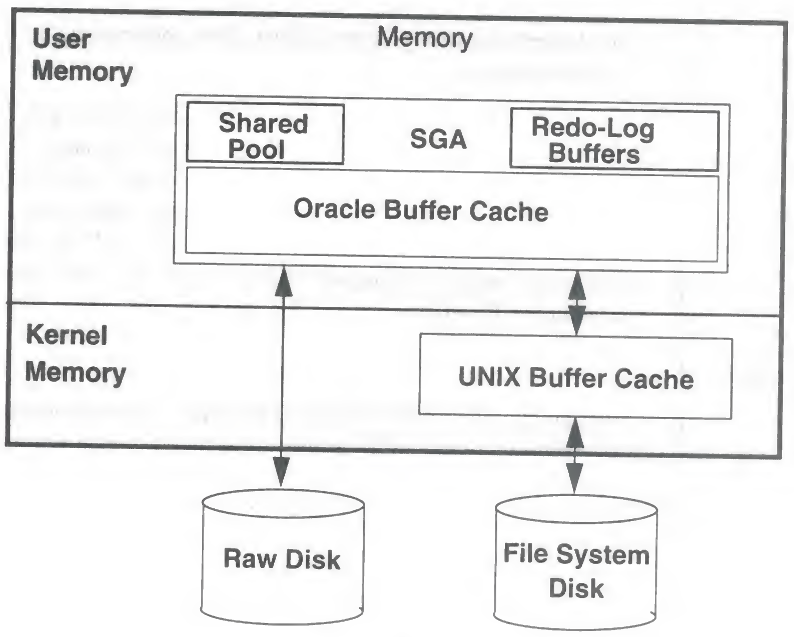


Figure 5: Oracle Buffer Cache and UNIX Buffer Cache

Figure 5 illustrates memory components of a UNIX system running Oracle. By maintaining data in memory, the UNIX kernel reduces disk I/O activity. Most of the tunable Oracle memory structures are components of the System Global Area (SGA), which resides in shared memory in user space, external to the kernel. The SGA contains the Oracle buffer cache, shared pool, and redo-log buffers. The Oracle data buffer cache contains copies of recently accessed database blocks; each buffer corresponds to one database block. The shared pool stores the library cache with shared SQL and PL/SQL areas, data dictionary cache, and session information. The redo-log buffers hold redo-log information. Tuning is accomplished by properly setting the size of these three SGA components to use enough, but not too much, memory. The size of the SGA is the sum of the size of these components plus some overhead items, and is computed for you during Oracle start-up.

The Oracle buffer manager ensures that more frequently accessed data is cached longer (and is therefore available without the need for an I/O request). Monitoring the buffer manager and tuning the buffer cache can have a significant influence on the performance of Oracle. Some of the following tips discuss ideas for tuning data and shared pool buffers. The optimal Oracle buffer size for your system will depend on the overall system load and on the relative priority of Oracle over other applications; if you increase the size of an Oracle buffer to improve Oracle performance, you may affect other applications by reducing the amount of available memory.

As Figure 5 illustrates, the relative size of the UNIX and Oracle buffer caches is also important since they both have to share the same limited system memory resources. When the UNIX file system is used, UNIX will read data from disk into the UNIX buffer cache (in system space) and then from the UNIX buffer cache to the Oracle buffer cache (in user space). When raw devices are used, the UNIX file system is bypassed. Consequently the UNIX buffer cache can be minimized (see Tip 40).

Tip No. 12: Look for Swapping

Swapping is one of the UNIX mechanisms to accommodate the size limitation of memory by moving entire processes to disk to reclaim memory.

```
% vmstat -S 5 9


 procs    memory                      page                          disk          faults     cpu
 r b w avm      fre   si so pi po    fr de sr x0 x1 x2 x3  in  sy  cs us sy id
 0 0 0   0      256    3  0  0  0     0  0  3  1  0  1  0 143 148 180  2 19 79
 0 0 0   0      608   15  0  0  0     0  0  0  0  0  4  1 269 446 243  9 33 58
 1 1 0   0      840    6  0  8  0     0  0  0  7  0  1  4 250 317 272  6 33 61
 1 0 0   0      744    9  0  0 32    32  0  0  9  0  0  0 282 496 328 15 48 37
 0 1 0   0      592    0  0  0 32    32  0  0 14  0  1  0 258 382 330  4 29 67
 0 0 0   0      704    0  0  0  0     0  0  0  4  0  6  1 233 296 291  6 34 60
 8 0 0   0      608    3  0  0  0     0  0  0  9  0  9  3 290 334 377  5 35 60
 0 0 0   0      472    3  0  0  0     0  0  0  0  0  2  1 222 219 306  3 26 71
 1 0 0   0      424    3  0  0  0     0  0  0  0  0  0  0 331 465 373 10 35 55
```

Figure 6: Sample vmstat Swapping Activity Display

Since swapping incurs significant UNIX overhead, it is important to monitor the status of the processes and take appropriate action when necessary. The utilities for checking if any Oracle background process is swapped out are *sar -w* on System V or *vmstat -S* on BSD UNIX. Your goal is to minimize swap-outs as much as possible. Figure 6 shows a sample output from *vmstat*. The column *w* indicates the number of runnable processes that have been swapped out (written to disk). If this value is nonzero, swapping is occurring and your system is having a serious memory shortage problem. The columns *si* and *so* indicate the number of swap-ins and swap-outs per second, respectively. Swap-outs should always be zero.

If your system is swapping and you need to conserve memory, first, don't run unnecessary system daemon processes or application processes. Second, decrease the database buffers to free some memory if possible. Depending on the platform, you can also decrease the number of UNIX file buffers especially if raw partitions are being used (see Tip 40 about using raw partitions).

Depending on your initial configuration, your system resource limitations, and your overall system load, the performance benefit that you might see by eliminating most swapping is roughly 0 – 100%.

Tip No. 13: Look for Paging

Paging is a UNIX mechanism to manage the limitation of memory. Unlike swapping where entire processes are moved, paging moves only individual pages of processes to disk to reclaim their memory. Paging algorithms try to maintain pages that were used recently in memory.

```
sar -p 10 10

dvlseq dvlseq 3.2.0 V1.4.0 i386     01/28/93

22:05:38      vflt/s     pflt/s     pgfil/s     rclm/s
22:05:48      173.37       0.00       0.00        0.00
22:06:08      411.08       0.00       0.00        0.00
22:06:18      172.83       0.00       2.63        0.00
22:06:28      827.02       0.00       0.20        0.00
22:06:38      350.20       0.00       0.00        0.00
22:06:48       11.72       0.00       0.00        0.00
22:06:58      412.73       0.00       1.31        0.00
22:07:08      515.83       0.00       0.00        0.00
22:07:19      511.60       0.00       0.00        0.00

Average       406.84       0.00       0.41        0.00
```

Figure 7: Sample *sar -p* Paging Activity Display

Paging may not present as serious a problem as swapping since the entire program doesn't have to reside in memory to run. A small amount of page-outs may not noticeably affect the performance of your system. However, the performance may degrade rapidly as page-out activity increases. Thus, it is a good idea to try to minimize page-outs as much as possible. It is also important to remember that page-outs may not cause serious performance degradation if you have fast CPUs and/or disks.

If you have a Sun, IBM AIX, or DG-UX system, significant paging activity is normal. Sun aggressively frees up memory by pre-paging. IBM and DG can use "free" memory for read-ahead file buffering. Detecting excessive paging on these systems will require running measurements during periods of good response or idle time to obtain a baseline for comparing against measurements during periods of sluggish response

The tools we can use to monitor paging are *sar -p* on System V and *vmstat -S* on BSD UNIX. Figure 7 shows a sample output from *sar*. The *vflt/s* column indicates the number of address translation page faults; address translation faults occur when a process references a valid page not in memory. If processes frequently reference paged-out memory, your system is experiencing a memory shortage. You need to learn from experience what typical address translation fault rates are on your system, and try to keep them low enough to provide good response time. Another important column in Figure 7 is *rclm/s*; the number of valid pages that have been reclaimed and added to the free list by page-out activity. This value should be zero.

If you are constantly having excessive page-out activity, the only solutions may be to get more memory, to off-load some of the work to another system, or to configure your kernel to use less memory. Although there are some ways to conserve memory, they are beyond the scope of this guide. UNIX system performance tuning books, such as those in the Bibliography, offer guidance on memory management.

If you can't add more memory or drop any applications to conserve memory, it is important to consider the tradeoffs in changing the size of the SGA. Suppose you would like to maintain the number of users you currently have. If you make the SGA too large, less memory will be available for other Oracle or non-Oracle processes and swapping and paging activities will increase. On the other hand, if you make the SGA too small, more memory will be available for other Oracle or non-Oracle processes but database I/O will increase.

If possible, you want the entire active portion of the database to be able to fit inside the buffer cache. However, in practice, you are likely to have limited memory and other activities on your system. Therefore you need to optimize the SGA size.

Tip No. 14: Allocate Enough Swap Space

A UNIX system uses swap space on disk to hold pages of process memory that have been paged/swapped out. Since a shortage of swap space might cause symptoms such as system hanging, slow response time, and unsuccessful spawning of new processes, it is important to ensure that your system has enough swap space. Unless your system supports file system paging (SVR4, HP-UX), changing the size of existing swap partitions may involve an extensive reorganization of your file system.

Start with swap space 2 - 4 times your system's physical memory size (more if you plan to use CASE, Oracle Applications, or Oracle Office), monitor your use of swap space, and increase it as necessary. To see how much swap space is in use, use *pstat -s* on BSD UNIX and *swap -l* on SYSTEM V. Procedures for adding swap space vary widely between UNIX implementations. Consult your system documentation for the procedure that applies to your system.

Tip No. 15: Control Paging

By default, the UNIX kernel will move memory pages to disk when physical RAM becomes constrained. For many applications, this is appropriate behavior. Sometimes, however, Oracle performance can be improved by exercising more control over the use of physical memory. The control provided by UNIX is very platform specific. By guaranteeing that the SGA is available despite heavy demands for memory from other programs, this tip can sometimes allow a significant increase in the number of users.

Some vendors who allow more memory control are:

- Data General -- DGUX has the *PERCENTBUF* kernel parameter.
- IBM -- AIX provides the *vmtune* utility to adjust *MINPERM* and *MAXPERM*.
- Sequent -- Use *vmtune* to adjust *minRS* and *maxRS*.

Tip No. 16: Hold the SGA in a Single Shared Memory Segment

Another consideration for memory tuning involves shared memory. A single shared memory segment should be big enough to hold the SGA. If there isn't enough shared memory available, you can't start up the Oracle instance. You may need to reconfigure the UNIX kernel to increase shared memory. The UNIX kernel parameters for shared memory include SHMMAX, SHMMNI, and SHMSEG. See the platform-specific *Installation Guide* for a recommended value of each parameter. To check the components of the SGA and their corresponding sizes, use the *Server Manager* Instance/Database selection or SQL*DBA:

```
>show sga
Total System Global Area        4408616 bytes
              Fixed Size          52120 bytes
           Variable Size        3938704 bytes
```

```
Database Buffers        409600 bytes
    Redo Buffers          8192 bytes
```

Use the Oracle utility *tstshm* (where available) to evaluate the existing shared memory configuration. This tool provides some useful information:

- The number of segments of shared memory generated and attached to the process
- The location of shared memory in virtual memory
- The size of shared memory
- The size of the largest single segment

Refer to the *Installation & Configuration Guide* for more information. You can also use the UNIX utility *ipcs* to monitor the status of shared memory.

Depending on your initial configuration, your system resource limitations, and your overall system load, the performance benefit that you might see is roughly 0 – 1%. Although this performance gain is minor, remember that you can't start the database without enough shared memory configured.

Tip No. 17: Lock the SGA in physical memory

The primary function of the SGA is to cache database information. If the SGA area begins to page to disk, then the caching becomes an overhead, rather than a benefit. Some platform vendors provide techniques to lock the SGA into physical memory.

While locking the SGA to physical memory can improve Oracle performance by from 0 to 20%, it may well reduce the performance of other applications on the same system. If a system will be used primarily or exclusively to run the Oracle Server, this may well be worthwhile. If the Oracle Sever is run along with many other programs, then Tips 12 and 13 will balance performance across all the programs.

Some vendors which allow user control of SGA locking are:

- ATT GIS - The *SHM_NAILED_GID1* parameter of */etc/conf/cf.d/stune* locks shared memory
- Data General -- DGUX has the *PERCENT LOCKABLE* kernel parameter.
- Sequent -- Use *vmtune* to adjust *DIRTYHIGH*, which frees physical memory for the SGA.

Tip No. 18: Make Oracle Block Size a Multiple of the O/S Block Size

The Oracle Server manages the database files in units usually called blocks (sometimes called pages). Database blocks refer to blocks which correspond to the Oracle block size; they may differ in size from the standard I/O block size of the host operating system. Since a UNIX system always reads entire O/S blocks from the disk, I/O bandwidth is used inefficiently if the database block size is smaller than the UNIX file system buffer size.

Your database block size should be equal to, or a multiple of your operating system block size. The database block size is set by the `initdb<n>.ora` parameter DB_BLOCK_SIZE. The block size can only be changed by recreating the database. The table below shows default Oracle block size, number of extents possible with that block size, and platforms that use that block size as the default.

Block Size	Number of Extents	Platforms that use size. (Most can also be set to a larger size.)
2K	121	Most UNIX, Sun, HP, Sequent
4K	249	IBM-AIX, Sequent (for large databases)
8K	507	(used for very large databases (VLDB))
16K	1017	(for VLDB's used for decision-support)

The default value of the DB_BLOCK_SIZE parameter is listed in your *Oracle7 for UNIX Installation & Configuration Guide.* You can also determine its value by querying the V$PARAMETER data dictionary table. The DB_BLOCK_SIZE parameter also determines the size of the database buffers in the System Global Area (SGA). The parameter determining the number of those buffers, DB_BLOCK_BUFFERS, is the parameter having the most direct effect on system performance, as discussed below.

Depending on your initial configuration, your system resource limitations, and your overall system load, the performance benefit that you might see is roughly 0 – 5%.

Tip No. 19: Optimize Number of Database Buffers

The Oracle *buffer cache* is the area in the SGA that holds copies of database blocks. These blocks contain data that is frequently accessed by Oracle processes. The number of buffers in the cache is determined by the `initdb<n>.ora` parameter DB_BLOCK_BUFFERS. The buffer cache reduces the time necessary to access data.

First, check the *hit ratio* through *Server Manager* or use *SQL* as follows:

```
SQLDBA> select name, value from v$sysstat where name in ('db block gets',
              'consistent gets', 'physical reads');
```

The *Hit Ratio* for the buffer cache is defined as:

$$\text{HitRatio} = \frac{\text{LogicalReads} - \text{PhysicalReads}}{\text{LogicalReads}}$$

where

LogicalReads = db block gets + consistent gets

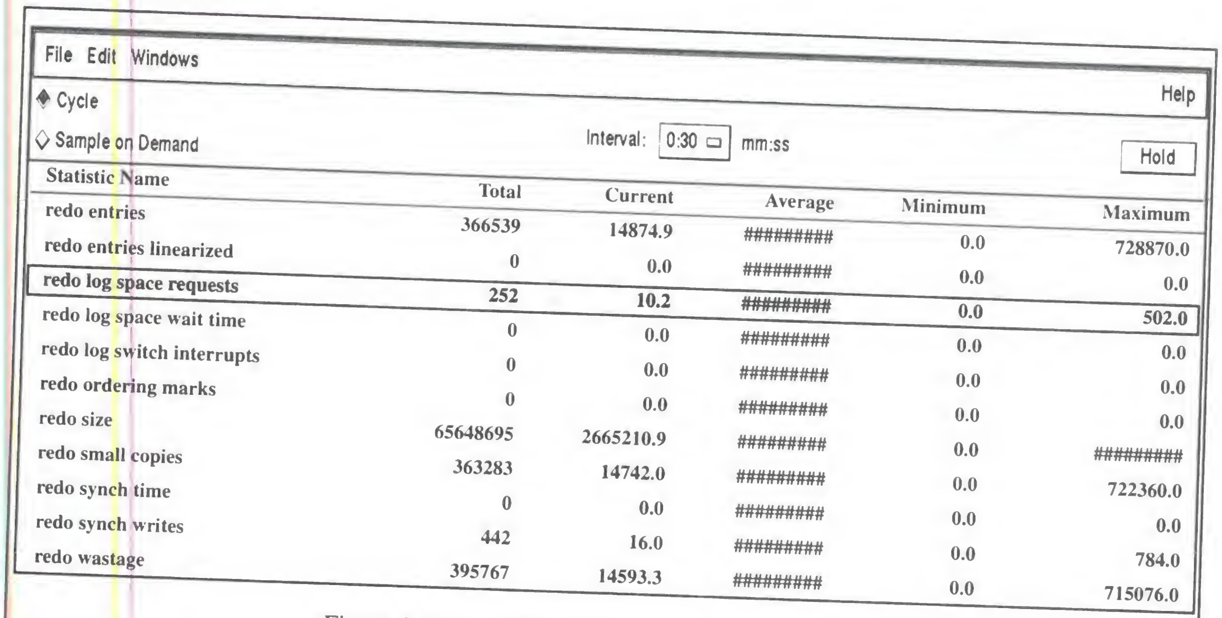
If your hit ratio is less than 60% or 70%, you may want to increase the number of buffers in the cache to improve performance by increasing DB_BLOCK_BUFFERS. (This system would benefit greatly from more buffers.)

The *Oracle7 Server Tuning Guide, Chapter 10*, describes how to use the table X$KCBRBH to predict how many buffers to add or remove based on statistics gathered on a running system. However, many experienced tuners observe the performance of the database buffers as described above, and adjust the number of buffers accordingly.

Depending on your initial configuration, your system resource limitations, and your overall system load, the performance benefit that you might see is roughly 0 - 200%.

Tip No. 20: Optimize Number of Redo Buffers

The *redo log buffer* is a circular buffer in the SGA that holds information about changes made to the database. As changes are made to the data in the database, those changes are logged in the redo log buffer. The *log writer process* (LGWR) is responsible for writing the redo log buffer to a redo log file on disk to ensure that space is always available in the buffer for new entries. Still, under heavy load, there may be contention for space in the redo log buffer.



Statistic Name	Total	Current	Average	Minimum	Maximum
redo entries	366539	14874.9	#########	0.0	728870.0
redo entries linearized	0	0.0	#########	0.0	0.0
redo log space requests	252	10.2	#########	0.0	502.0
redo log space wait time	0	0.0	#########	0.0	0.0
redo log switch interrupts	0	0.0	#########	0.0	0.0
redo ordering marks	0	0.0	#########	0.0	0.0
redo size	65648695	2665210.9	#########	0.0	#########
redo small copies	363283	14742.0	#########	0.0	722360.0
redo synch time	0	0.0	#########	0.0	0.0
redo synch writes	442	16.0	#########	0.0	784.0
redo wastage	395767	14593.3	#########	0.0	715076.0

Figure 8: Server Manager System_Statistics Display

You can monitor through *sqldba* the statistic *redo log space requests* which reflects the number of times a user process waits for space in the redo log buffer:

```
sqldba> monitor systemstatistic Redo
```

Figure 8 shows an equivalent example in *Server Manager*. The *Total* value of redo log space requests should be near 0, or not increasing. A nonzero value (as is the case in Figure 8) indicates that processes are waiting for space in the buffer. In this case, consider increasing the size of the redo log buffer (in increments of 5%). The size of the redo log buffer is determined by the `initdb<n>.ora` parameter LOG_BUFFER whose value is expressed in bytes.

Tip No. 21: Optimize the Shared Pool Size

The shared pool in the SGA holds the library cache with shared SQL and PL/SQL areas, data dictionary cache, and session information. All three functions compete for space in the shared pool. Session information is stored only if the multi-threaded server architecture is used. The modified least-recently-used (LRU) algorithm gives precedence to data dictionary cache entries. This means that tuning the library cache will also ensure that enough memory is available for the data dictionary. The `initdb<n>.ora` parameter SHARED_POOL_SIZE sets the size of the shared pool in bytes.

Use V$SGASTAT to monitor the shared pool. Watch the free space. The example below shows a V$SGASTAT query.

```
>SELECT * FROM v$sgastat ORDER BY bytes desc
NAME                           BYTES
-------------------------- ----------
sql area                      1370876
free memory                    867036
library cache                  785224
db_block_buffers               409600
dictionary cache               275740
 . . .
```

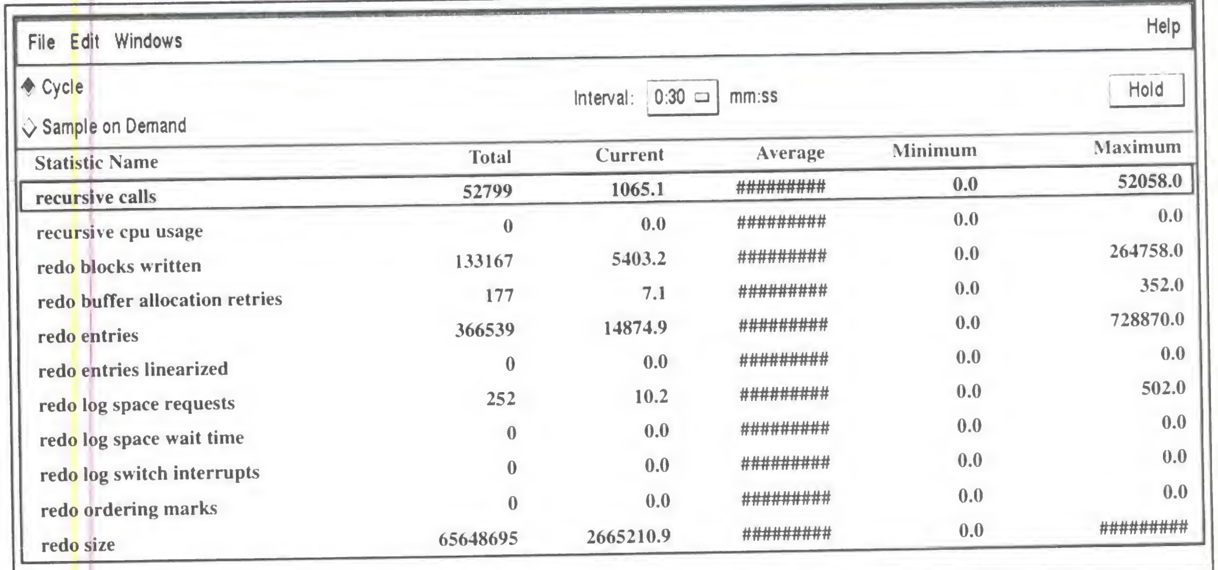
Our tuning experts report that the shared pool is often set too large. If the free memory area is as large as the example above, consider reducing the size of the shared pool. Also look to see if one of the values is increasing. If so, use the techniques identified in the other tips to determine the cause.

Tip No. 22: Verify Data Dictionary Cache Effectiveness

The data dictionary is a collection of database tables and views containing information about the database schema, its structures, and its users. Since most SQL statements access tables and fields, information in the data dictionary is accessed frequently by the Oracle Server.

For best performance when parsing SQL statements, the data dictionary cache must be large enough to hold the most often accessed data. Data dictionary cache misses will generate recursive calls and affect the performance of your database. Recursive calls are SQL statements issued by the Oracle Server itself when executing SQL statements. An indirect way to determine whether there are data dictionary cache misses is by monitoring the number of *recursive calls* through either *Server Manager* or *sqldba System Statistics User.*

Figure 9 displays a sample output from *Server Manager.* The *Total* column shows the cumulative value for *recursive calls* since you last started the database. In Figure 9, you can see that the total value for the *recursive calls* is 52799. If the Oracle Server does not continue to make recursive calls after start-up, your data dictionary cache is probably large enough for your dictionary data. If the number of recursive calls accumulates while your application is running, then there may be data dictionary cache misses. Since stored procedures also cause *recursive calls*, you should query the V$ROWCACHE table to check the cache activity if you suspect that your dictionary cache may be too small. Figure 9 shows a system that may have a shared pool size that is too small, since the Oracle Server being monitored has been running a short time and uses no stored procedures.



◆ Cycle
◇ Sample on Demand
Interval: 0:30 mm:ss

Statistic Name	Total	Current	Average	Minimum	Maximum
recursive calls	52799	1065.1	#########	0.0	52058.0
recursive cpu usage	0	0.0	#########	0.0	0.0
redo blocks written	133167	5403.2	#########	0.0	264758.0
redo buffer allocation retries	177	7.1	#########	0.0	352.0
redo entries	366539	14874.9	#########	0.0	728870.0
redo entries linearized	0	0.0	#########	0.0	0.0
redo log space requests	252	10.2	#########	0.0	502.0
redo log space wait time	0	0.0	#########	0.0	0.0
redo log switch interrupts	0	0.0	#########	0.0	0.0
redo ordering marks	0	0.0	#########	0.0	0.0
redo size	65648695	2665210.9	#########	0.0	#########

Figure 9: Server Manager Statistics Display

Tip No. 23: Allocate enough Library Cache

The library cache contains shared SQL and PL/SQL areas. Even if SQL can be reused (Tip 8), it will not be if the library cache is too small. If an application makes an execute call for a SQL statement and its parsed form has been deallocated from the library cache, Oracle must reparse it taking a significant amount of time. Determine if these misses are affecting performance by querying the table V$LIBRARYCACHE.

Monitor the statistics in the V$LIBRARYCACHE over a period of time with this query:

```
SELECT SUM(pins) "Executions",
       SUM(reloads) "Cache Misses while Executing"
    FROM v$librarycache;
```

Giving a possible output of:

```
Executions Cache Misses while Executing
---------- ----------------------------
    320871                          549
```

The sum of the PINS column indicates that SQL statements, PL/SQL blocks, and object definitions were accessed for execution a total of 320, 871 times. The sum of the RELOADS indicates that 549 of those executions resulted in library cache misses. Total reloads should be near zero, and the ratio should be below 1%.

If the ratio of RELOADS to PINS is greater than 1%, allocate additional memory to the library cache by increasing the `initdb<n>.ora` parameter SHARED_POOL_SIZE. Proper adjustment of the library cache has resulted in performance gains of up to 50%.

Tip No. 24: Lock large PL/SQL blocks into the shared pool

Sometimes semi-frequently used shared objects should be locked into the shared pool. Objects are loaded into a contiguous block of memory in the library cache. If a large PL/SQL package is aged out of the shared pool and then reused, sometimes it is necessary to age out many more objects than their combined size might suggest to generate enough contiguous space for the reloaded object. During the time needed to free up the space and (re)load library cache, the library cache latch is held exclusively, and all potential users of the library cache must wait. Of course, all the SQL and PL/SQL that was aged out must be reparsed and reloaded before it can be used. If the library cache latch is often seen as a bottleneck, then this tip can often help. (Note: this tip is valid for 7.1 and 7.2 Oracle. Futures versions of Oracle may change this functionality.)

Use the *dbms_shared_pool* utility package to discover the size of objects in the shared pool. (Documentation is in the comments of the pl/sql script *dbmspool.sql* normally installed at $ORACLE_HOME/rdbms/admin.) The following procedure installs the PL/SQL, enables the server output buffer, and runs the sizes() procedure.

- Navigate to $ORACLE_HOME/rdbms/admin.
- Invoke sqlplus or sqldba and execute the following sequence of commands:

```
@dbmspool
@prvtpool
set serveroutput on size xxxxxx
begin
sys.dbms_shared_pool.sizes ( minsize number );
end;
/
```

A setting of 20000 should be big enough for xxxxxx, and 20 a starting number for *minsize*. Once the large objects that are semi-frequently used are identified, the procedure *keeping* varchar2, flag char DEFAULT 'P') can be run as many times as needed to lock objects into the shared pool. (You may want to build a splplus script to do this as part of start-up.) The procedure *unkeep* is used to unlock objects.

Tip No. 25: Optimize the session cache cursors

If you are using Oracle Forms applications extensively, or other programs that close and then reopen session cursors frequently, asking the Oracle Server to cache session cursors can make a big difference in performance. Set the initialization parameter SESSION_CACHED_CURSORS to the maximum number of session cursors to keep in the library cache. Monitor the performance of the session cursors cache and adjust cache size based on the hit ratio. SQL similar to the example below retrieves data for tuning caching session cursors (see 9-10 of the *Oracle7 Server Documentation Addendum, Release 7.1.*)

```
> SELECT value, name
    FROM V$sysstat WHERE statistic# IN (122, 123)

VALUE      NAME
---------- ----------------------------
     12675 session cursor cache hits
     12766 session cursor cache count
2 rows selected.
```

IX. Step 5: Tuning Disk I/O

I/O bottlenecks are often the easiest performance problems to identify. In this section, you will learn how to use *sar*, *iostat*, and *vmstat* to identify disk devices that are potential I/O bottlenecks, and how to distribute the files causing the problem onto other disk devices. These tips will help you balance I/O evenly across all available disks and reduce disk access time.

Before proceeding to the following steps, minimize the potential for internal Oracle I/O bottlenecks, e.g., put redo logs on their own disks and separate indexes from tables. More information is available in the references described in the Bibliography.

Tip No. 26: Place Redo Logs on their Own Disk Device

If your Oracle applications involve heavy INSERT and UPDATE activity, you can maximize Oracle performance by locating your redo logs on disks that support no other disk activity. Also, if you have enabled the ARCHIVING option, place each redo log on a separate disk to minimize disk contention between the LGWR process (writing to the current redo log) and the ARCH process (reading from the closed redo log).

Placing redo logs on raw (character special) files can enhance performance further. Redo logs should be the among the first files to be put on raw devices for the following reasons.

- Redo files are written and read sequentially, maximizing the benefits of raw devices.
- Asynchronous read and write is likely to be available for raw devices.
- Redo files do not need to be backed-up if your site is using an off-line backup strategy, minimizing raw device administrative costs, and
- The size of redo files is fixed, minimizing raw device administrative costs.

Depending on your initial configuration, your system resource limitations, and your overall system load, the performance benefit that you might see is roughly 0 – 15%.

Tip No. 27: Balance Disk I/O across all disk drives

In general, ensuring that I/O is balanced across every disk drive is an essential part of tuning your Oracle Server. For smaller databases and those not using the parallel query option, ensuring that different datafiles and tablespaces are distributed across the available disks may be sufficient. Tips 32 - 35 show how to tune by moving files around. Appendix 1, OFA, describes a mechanism and naming conventions designed to facilitate balancing the I/O load.

The three tips below (28 - 30) describe how to balance the datafiles belonging to a single tablespace across as many disks as necessary. While unneeded for smaller systems, this tuning operation is essential for best results for very large databases and effective use of the parallel query option. If available, use a logical volume manager for first level striping. You can then stripe across multiple logical volumes if necessary.

Tip No. 28: Balance Disk I/O with a Logical Volume Manager

Many operating systems now include a Logical Volume Manager(LVM) as part of their standard distribution. A LVM can be used to stripe data across multiple disk drives in order to distribute I/O across multiple disk spindles. A typical stripe size is 4 Mb, but trial and error is usually used to find the best size. In general, smaller stripes are better for OLTP with random access, while bigger stripes are better for sequentially accessed data often encountered in decision support applications or when using the parallel query option. Redo logs should also use large stripes or no stripes. The stripe size should be a few times larger than the `initdb<n>.ora` parameter DB_FILE_MULTIBLOCK_READ_COUNT times your DB_BLOCK_SIZE. Be sure to stripe data files containing indexes and clusters as well. Performance gains vary widely depending on the size of the tables and the efficiency of the LVM. When using the parallel query option to do full table scans of large tables, LVM can produce performance gains of 50% to over 500% compared to unstriped tables.

Some of the vendors who provide LVM's include:

- Hewlett Packard -- HPUX provides drive stripes as small a 1MB, and is reported to give "near RAID" performance.
- Data General - DGUX provides a LVM which allows system memory to be dynamically configured as cache for the drives.
- IBM - AIX provides mirrored logical volumes, which improves read performance.
- Pyramid Technology -- DYNIX/ptx allows blocks as small as 2k.
- Novell -- UNIXware provides a GUI based LVM.

Tip No. 29: Balance Disk I/O using the DATAFILE keyword.

Oracle also allows data files to be striped without a LVM. This is done with the DATAFILE keyword of the CREATE TABLE command. Performance will usually be better with a LVM. A LVM encourages a smaller stripe size, which tends to distribute I/O more randomly and more automatically. If a LVM is available, use it rather than the DATAFILE keyword. However, careful use of the DATAFILE keyword can improve parallel queries by 50% to 500% over results from a large, unstriped table.

Tip No. 30: Use RAID devices

RAID can provide striping of data across multiple disk drives. To some extent, RAID has gotten a bad reputation -- particularly when an expensive RAID device has been added to a system and no performance increase is seen. While RAID technology can increase the performance of reads like the other striping methods (Tips 28 and 29), writes will slow down for most uses of RAID. This is because a write must send data to two disk drives rather than one. However, the "extra" write gives RAID more functionality than just performance boost.

There are three common reasons to add RAID devices:

1. Improve reliability. (Typically RAID 1 (mirroring) or RAID 5)
2. Improve read response. (Typically with built-in mirroring RAID)
3. Improve performance through distributed I/O.

The first two of these are achieved by choosing the RAID device which has those features. Improving performance, however, is much more difficult to predict. There are many variables to consider when choosing a RAID device. A few are: external bus, internal bus, number of busses, external cache, internal cache, drive types, drive caches, internal read/write algorithms, and the Oracle distribution of write requests and write sizes.

RAID vendors are aware of the performance issue. Most have added large, battery backed-up caches to their controllers. To a large extent, this provides an improvement in performance even when the application is not well suited to the particular RAID device.This allows the internal controller algorithm to optimize I/O for the cached data, rather than relying exclusively on Oracle's I/O distribution to match the particular RAID design.

RAID is typically used to hold all Oracle data files, including tables, indexes and clusters, since those tend to be accessed randomly. The rollback logs are also often stored on RAID devices, since they are typically shared between users, and therefore have some degree of random access. Redo log files, however, should not be put on RAID devices. Redo Log files are accessed sequentially, and benefit from being placed on a fast, lightly loaded drive. Oracle's tuning specialists often stripe files across multiple RAID sub-systems, using the RAID for reliability, then striping across RAID's for parallel performance.

Tip No. 31: Tune the Database Writer to increase write bandwidth

Updates and Inserts to a database must eventually be written to disk. For the Oracle server, a single logical process called the *Database Writer* (DBWR) writes all changed database blocks to disk. If this logical process is contained in a single UNIX process using a normal synchronous disk write call, it can only do one write at a time. If the system is doing many inserts or updates, this serializing of database writes can become a bottleneck.

Oracle offers three different technologies to prevent DBWR activity from becoming a bottleneck: List I/O, Asynchronous I/O and use of Multiple DBWRs. Before using any one of these techniques, determine if your system needs more DBWR bandwidth. For current status, Query the V$SESSION_WAIT view (or V$SYSTEM_EVENT view for status since start-up). If the event *free buffer waits* shows time spent (see below), it means that the DBWR was a bottleneck. If this value is significant, consider applying one of the remedies below.

```
>select * from v$system_event;
 EVENT                    TOTAL_WAIT
-----------------         ----------
. . .
free buffer waits              463
. . .
```

If your database is built on the filesystem and not raw devices (Tip 40), be sure to use a method that is compatible with file system I/O. Depending on your initial configuration, your system resource limitations, and your overall system load, the performance benefit that you might see is roughly 0 – 15%; however, for some write intensive operations like parallel index build, lack of DBWR bandwidth can be an overwhelming bottleneck.

Use List I/O

Many SVR4 releases provide a List I/O feature. This allows many I/O requests to be put into a "list," which is treated as a single I/O request. This decreases the number of UNIX context switches which are required to perform disk I/O. List I/O provides "non blocking" writes, which allow your program to continue execution after executing a write. See your *Installation and Configuration Guide* for current status information. If available, use list I/O rather than multiple DBWRs.

Some of the vendors who provide List I/O include:

- HPUX -- List I/O for raw files
- IBM AIX -- List I/O for raw and filesystem files
- NCR -- After loading from tape, the kernel can be rebuilt to include List I/O
- Novell -- UnixWare provides List I/O for raw files now, filesystem files soon, enabled/disabled with *initdb<n>.ora*
- SGI -- List I/O for both raw and filesystem files
- UNISYS -- List I/O is provided on some of their models

Use Asynchronous I/O

Asynchronous I/O is an I/O mechanism that allows processes to proceed with the next operation without having to wait after issuing a write. Asynchronous I/O improves the system performance by minimizing the wasted idle time; DBWR won't be blocked on each I/O.

Use asynchronous I/O if it is available on your system. It can be enabled by setting the parameter ASYNC_WRITE (or USE_ASYNC_IO on some platforms) to true in `initdb<n>.ora`. Asynchronous I/O is available on many Oracle for UNIX platforms but may require the use of raw disk devices or special kernel configuration. Consult your *Installation and Configuration Guide* for more information. If your database files are not on raw devices, multiple DBWRs may be a better choice than converting to raw device use to gain the benefits of asynchronous database writes.

- Sun Solaris -- Async I/O for both raw and filesystem files
- SCO -- Async I/O for raw files
- Pyramid -- Async I/O for raw files
- Sequent -- Async I/O for raw files

Use Multiple DBWRs

If free buffer waits are significant, consider using multiple DBWRs. The parameter for this is DB_WRITERS in `initdb<n>.ora`. Since DBWRs are sometimes assigned to write to the same disk, you may need more than one DBWR per disk. Start conservatively, and measure the results. Adjust as necessary. Oracle tuning specialists report seeing too many DBWR's, rather than too few, at several recent tuning engagements.

Tip No. 32: Look for Large Disk Request Queues

A request queue indicates how long the I/O requests on a particular disk device must wait to be serviced. Request queues can be caused by a high volume of I/Os to that disk or by I/Os with a long average seek time. Ideally, disk request queues should be at or very near zero. If the request queue is too large, then you have too many I/Os waiting for access to the same disk device, and this can create a performance bottleneck.

Use *sar -d* on System V and *iostat* on BSD UNIX utilities to analyze the disk activity. Figure 10 presents a sample report from *sar* on our untuned system. The column *avque* indicates the average number of requests outstanding during that time. Look for disks that maintain a large average request queue; a value consistently greater than 2 to 4 should warrant some attention. A small disk request queue doesn't necessarily mean that disk is not busy; you need to look at both *avque* and *%busy* (60-70% or more for *%busy* indicates very high disk activity). An example might be disks that write only redo logs; they are sequentially written and don't require seek times.

A large request queue is sometimes unavoidable, especially if all I/O accesses are being made to the same file. However, if the request queues are caused by access to multiple files on the same disk, you should reduce the queues by distributing some of the hot files to other disks. (See the following tips.)

Although the report from *iostat* doesn't include disk request queues, it shows which disks are busy. This information is valuable when you need to balance I/O workload in the following tips. In Figure 11, the field *bps* indicates the number of kilobytes transferred per second and the field *tps* indicates the number of transfers per second.

In order to correctly interpret the output of *sar* or *iostat*, you need to find out the throughput of your disk drives as installed on your system. In theory, one way to

```
% sar -d 5 2

dvlseq dvlseq 3.2.0 V1.4.0 i386     01/12/93

18:11:33   device   %busy   avque   r+w/s   blks/s   avwait   avserv

18:11:38     zd0       74     3.7      25      288     78.1     29.2
             zd1       10     1.2       4       42      5.5     26.6
             zd3        2     1.0       1       13      0.0     25.5
             zd4      102    10.5      39      493    246.7     26.1

18:11:43     zd0       27     4.1      11      131     74.1     23.8
             zd1        4     1.3       2       18      7.1     28.1
             zd3        1     1.0       0        6      0.0     26.0
             zd4      102    12.8      43      518    277.1     23.6

Average      zd0       50     3.8      18      210     76.9     27.5
             zd1        7     1.2       3       30      6.0     27.0
             zd3        2     1.0       1        9      0.0     25.7
             zd4      102    11.6      41      505    262.7     24.8
```

Figure 10: *Sar -d* Disk Activity Display

```
% iostat 5 10

     tty          xy0            xy1             xy2             xy3            cpu
tin tout bps tps msps  bps tps msps  bps tps msps  bps tps msps  us ni sy id
  1   28   9   1 23.3    9   2 22.0   11   2 18.6    4   1 19.7   3  0 12 85
  3   57  17   3 24.1    7   1 17.4   25   4 16.0    2   0 16.0   8  1 37 55
  1   42  20   4 21.7    3   1 10.8   32   6 23.9   10   1 43.7   8  0 34 57
  3   52  73   9 24.0    3   0 41.0   51   6 25.1   21   3 29.9   8  0 35 58
  2   46   8   1 20.0    0   0  0.0    0   0  0.0    9   1 21.1   9  0 38 53
  1   43  46   7 18.0   11   3 17.9    0   0  0.0    0   0  0.0   3  0 28 69
  4   60   0   0 39.5    0   0  0.0    0   0  0.0    5   1 22.7   6  0 25 69
  0   33   5   1 22.7    0   0  0.0    0  12 14.0    0   0  0.0   8  0 33 59
  1   38   2   0 26.0    0   0  0.0   89  18 15.5    6   1 26.0   7  0 41 52
  2   63  64   9 20.9    2   0 96.0   68  11 19.6   10   1 47.7   9  0 51 40
```

Figure 11: *Iostat* Disk Activity Display

do this is to look up in your system manuals the capacity of your disk subsystem. As a practical matter, it is often more straightforward to run some simple tests on an unloaded system and calculate the throughput. A typical command which will allow simple timing is:

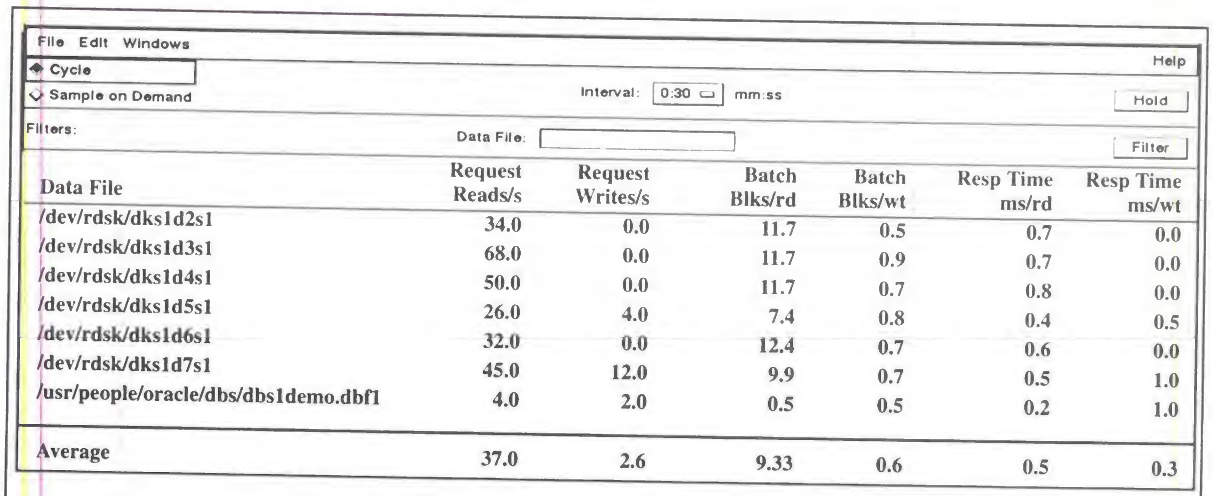
```
$ time dd if=/dev/rdisk1 of=/dev/rdisk2 bs=4096
```

This command will return the number of blocks transferred and the time to run the command. It is important to remember that normal UNIX operation will slow down the performance due to operations like context switching, interrupt servicing, and other processes on the system. Still, this command can give realistic performance numbers for an installed system, and can be expanded in a shell script to parallelize the test for multiple drives and controllers.

Tip No. 33: Identify Hot Files

"Hot" files are disk files with a high I/O rate. What constitutes a high rate will vary depending on the system or hardware you have. With disk access times currently at about 8 - 20ms, I/O rates approaching 50-70 I/Os per second would indicate a need for closer inspection. To examine disk access to Oracle files, use the *sqldba* monitor fileio or *Server Manager.*

Figure 12 shows a sample output from *Server Manager.* The *Server Manager* File I/O display lists all the database files accessed by the Oracle Server, along



Data File	Request Reads/s	Request Writes/s	Batch Blks/rd	Batch Blks/wt	Resp Time ms/rd	Resp Time ms/wt
/dev/rdsk/dks1d2s1	34.0	0.0	11.7	0.5	0.7	0.0
/dev/rdsk/dks1d3s1	68.0	0.0	11.7	0.9	0.7	0.0
/dev/rdsk/dks1d4s1	50.0	0.0	11.7	0.7	0.8	0.0
/dev/rdsk/dks1d5s1	26.0	4.0	7.4	0.8	0.4	0.5
/dev/rdsk/dks1d6s1	32.0	0.0	12.4	0.7	0.6	0.0
/dev/rdsk/dks1d7s1	45.0	12.0	9.9	0.7	0.5	1.0
/usr/people/oracle/dbs/dbs1demo.dbf1	4.0	2.0	0.5	0.5	0.2	1.0
Average	37.0	2.6	9.33	0.6	0.5	0.3

Figure 12: Server Manager File I/O Display

with the statistics reflecting their I/O activity. To identify hot files, compare the activity levels of your database files. For each database file, observe the statistics:

Request Read/s	Average number of reads from each database file per second
Request Write/s	Average number of writes to each database file per second
Batch Blks/wt	Average number of data blocks written to each database file in a single write

The total I/O rate for a single disk is the sum of *Request Read/s* and *Request Write/s* for all the Oracle database files managed by the Oracle instance on that disk. Determine this value for each of your disks and compare it to the I/O capacity of your disk. Also determine the total number of blocks written to each disk per second. For a single file, the number of blocks written per second is the product of *Request Write/s* and *Batch Blks/wt*. The total number of blocks written to each disk is the sum of the blocks written to all of its files. You can then compare disks to determine which disk has a high I/O activity.

You also need to observe the activities on disk through *sar*, *iostat*, or *vmstat -d* since disks holding database files and redo log files may also hold files that are not related to Oracle. Look at a sample output from "*sar -d*" in Figure 10 to examine the total I/O to your disks. The *r+w/s* column shows the number of reads and writes per second per disk. You can also check the *%busy* value; 60-70% or more indicates very high disk activity. Then try to reduce any heavy access to disks that contain Oracle files. You can also check physical I/O for each file by querying the V$FILESTAT or X$KCFIO table.

Tip No. 34: Move Hot Files to Other Disks

Distribute hot files to less active disk devices to balance disk I/O. You can move one entire hot file from a very active disk to a less active disk. You can also stripe a hot file across multiple disks so that part of the file is on each disk (Tip 27).

Tip No. 35: Reduce I/O to Hot Files

If there is only one hot file on a disk device and it is by itself responsible for the large request queue, moving it to another disk will not help. However, if that file is an Oracle database file, then consider one or both of the alternatives described below.

If the Oracle file or tablespace in question contains data from multiple segments (tables, indexes, etc.), move the heavily accessed segments to separate tablespaces and, hence, to separate files (remember that only at the tablespace level can a physical device for a database segment be specified). (See OFA Section IV in Appendix 1 for more information on tablespace naming and function.) If you have built an OFA compliant database (Tip 3), you already have much of the flexibility you need. You can also separate data segments and their corresponding index segments into separate tablespaces.

The steps for changing segment tablespaces are included in Appendix 3. The steps for separating data and index segments are described in Appendix 4. If only one segment is involved, consider table striping to place the segment data into multiple files in a single tablespace (Tip 27).

After you have redistributed the hot files, check whether you reduced your request queues. If not, you may have to examine the applications on your system to see if the I/O is unavoidable. More information is available in the references described in the Bibliography.

Depending on your initial configuration, your system resource limitations, and your overall system load, the range of performance benefit that you might see is from 0 to 50%.

Tip No. 36: Check for Excessive Database Fragmentation

The fragmentation of Oracle data structures requires the CPU to piece together the elements of a single logical I/O from multiple physical I/Os; the extra overhead degrades response time.

Extent fragmentation. Database segments may be composed of many non-contiguous extents of disk space. I/O times increase due to non-sequential disk reads and/or split I/Os. A split I/O occurs when a single I/O request must be split into two or more physical I/Os because the requested data spans non-contiguous extents on the disk.

Tablespace Fragmentation. Oracle tablespaces may be composed of many individual files. The Oracle segments (tables, indexes, etc.) within a tablespace may be composed of many individual extents, resulting in tablespace fragmentation. Sometimes this sort of fragmentation is desirable, as in the case of table striping, but in most cases it is not. Each time a database segment is dropped, it causes fragmentation in the segment's tablespace. Tablespace fragmentation can cause inefficient use of free space.

Tablespace fragmentation prevents Oracle from taking advantage of its multiblock read capability. A severely fragmented tablespace file can also waste usable database space if the segment extents are larger than the contiguous free extents.

Tablespace fragmentation can be identified from *Server Manager* or by issuing the following query from SQL*DBA:

```
SQLDBA> SELECT * FROM DBA_EXTENTS;
```

Free space can also be monitored from *Server Manager* or SQL*DBA:

```
SQLDBA> SELECT * FROM DBA_FREE_SPACE;
```

The Oracle*APS tool *dbmap* can also be used to check database fragmentation. It provides the information you get from the queries above in a well structured format. Figure 13 shows a portion of the *dbmap* output. It illustrates two types of free space fragmentation:

- Bubbling (block 8244) where small bubbles of noncontiguous free space form whenever an extent lying between active extents is dropped; and
- Honeycombing (blocks 9545, 9550, 9555, etc.) where free space is sectioned into contiguous pieces whenever adjacent extents are dropped.

A high *recursive calls* value in `utlbstat/utlestat` reports may also suggest table fragmentation, assuming the data dictionary cache has already been properly tuned. Appendix 2 has instructions for using `utlbstat/utlestat`.

```
  Tablespace     File  Block Id   Size                 Segment
--------------- ---- ---------- ------- -------------------------------
  SYSTEM            1          2      25     SYS.SYSTEM
                    1         27      25     SYS.SYSTEM
                    1         52      60     SYS.C_OBJ#
                    1        112       5     SYS.I_OBJ#
                              .
                              .
                              .
                    1      8,232      12     SYS.SAVE_ROLL
                    1      8,244       3     <free>
                    1      8,247     512     TP1.IACCOUNT
                    1      8,759     512     TP1.IACCOUNT
                    1      9,271       5     SYS.SAVE_STATS
                    1      9,276       5     SYS.SAVE_KQR
                    1      9,286     100     TP1.HISTORY
                    1      9,386      93     SYS.SAVE_STATS
                    1      9,479       5     SYS.I_OBJ1
                    1      9,484      41     SYS.C_OBJ#
                    1      9,525       8     SYS.I_OBJ2
                    1      9,533      12     SYS.I_XREF1
                    1      9,545       5     <free>
                    1      9,550       5     <free>
                    1      9,555       5     <free>
                    1      9,560       5     <free>
                    1      9,565       5     <free>
                    1      9,570       5     <free>
```

Figure 13: Sample dbmap Output

Tip No. 37: Eliminate Tablespace Fragmentation

At the simplest level, you can eliminate fragmentation by exporting table/tablespace data, removing and recreating the tablespace, and then recreating the table/tablespace with the proper storage parameters. However, this will not necessarily prevent the problem from recurring.

You can avoid tablespace fragmentation by sizing segment storage parameters properly (Tips 4, 5, and 6) and grouping segments with similar growth characteristics in the same tablespaces. Most importantly, you can minimize tablespace fragmentation by allocating all temporary segments in their own tablespace. (See OFA Rule 3.1 in Appendix 1.) Consult the *Oracle7 Server Administrator's Guide* for details.

Tip No. 38: Use More Database Buffers

If your system is I/O bound, increase the number of database buffers to cache more data and avoid I/O. Increase the hit ratio as long as it doesn't increase paging.

You will notice that this tip is closely related to Tip 19 ("Optimize Number of Database Buffers"). As we mentioned in the introduction to this guide, tuning a system for optimal performance is an iterative process: memory configuration may affect I/O rates and vice versa.

Depending on your initial configuration, your system resource limitations, and your overall system load, the range of performance benefit that you might see is approximately 0 to 10%.

Tip No. 39: Choose the best File System Type

Most UNIX systems allow a choice of file systems. File systems have very different characteristics, and the techniques they use to access data can have substantial impact on the performance of your database. Some typical file system choices are:

- *s5* - The UNIX System V File System
- *ufs* - The "UNIX File System," derived from Berkeley (BSD).
- *vxfs* - The Veritas File System
- raw device - No File System. See Tip 40.

There are many types of file systems, and their suitability to particular applications is usually undocumented. Even *ufs* file systems are hard to compare because their implementations differ. While *ufs* is often the high performance choice, performance differences vary from 0 to 20% depending on the file system chosen. Use the *mkfs* UNIX utility or a higher level *sysadm* type utility (if available) to install file systems on your system.

Tip No. 40: Use Raw Partitions/Devices (if I/O bound)

Using raw partitions instead of a file system can improve performance since the database writer bypasses the UNIX buffer cache and eliminates the file system overhead, which result in fewer instructions per I/O. (Because the number of disk accesses remains the same, raw I/O reduces the load on the processor(s), not the disk.)

The use of raw partitions can improve system performance, but it comes with significant cost. Raw partitions are not as flexible as file systems. You should avoid using the first partition as a raw partition on some platforms since it has some important system information that should not be overwritten. Furthermore, you can back up raw partitions only by using "*dd*" or the *Oracle Parallel Backup Restore Utility* (as it becomes available). It is also necessary for the DBA and system administrator to coordinate their work since raw partitions are not visible, unlike mounted file systems, unless symbolic links are used.

There are several other disadvantages of using raw partitions. It is harder to plan database configuration since you may not have a sufficient number of properly-sized raw partitions. It may also be impossible to move a data file from a "hot" disk drive to a "cool" drive for performance improvement since an acceptably sized section may not exist on a "cool" drive. It is still important to distribute I/O as much as possible to avoid a bottleneck, so several RAW partitions will normally be required.

If you are using raw partitions, minimize the UNIX buffer cache size since Oracle bypasses the UNIX file system. The UNIX kernel parameter for this is BUFPCT or NBUF. This is global so it may adversely affect other applications on the same system. See Tip 61 for important tradeoffs in changing the kernel parameters. See Appendix 1, Section V. for information on how to configure an Oracle database using raw I/O.

Depending on your initial configuration, your system resource limitations, and your overall system load, the performance benefit that you should see is between 5 and 40% for the same number of disk drives.

Tip No. 41: Redistribute Applications

If the above steps still do not eliminate I/O bottlenecks, consider moving some applications to another system.

Tip No. 42: Purchase More Disks

If none of the above is successful and your disk I/O rate is saturated, consider adding more disk drives and controllers to your system.

X. Step 6: Tuning CPU Usage

This section addresses some issues that involve analyzing CPU utilization to improve your Oracle performance. If your system is inappropriately or inefficiently configured, tuning

Oracle, disk, and/or memory parameters will not necessarily increase performance.

In theory, maximum system throughput occurs when all the system CPUs operate at 100% capacity; however, in practice, this isn't achievable. Application loads are very dynamic and load spikes make it impossible to maintain 100% utilization in a production system.

Tip No. 43: Balance CPU Loads

You can monitor CPU loads with *sar* on System V and *vmstat* and *iostat* on BSD UNIX. See Figure 14 for a sample output from *sar -u*. The CPU can be spending its time in different states: user (*usr*), system (*sys*), waiting for I/O (*wio*), and idle (*idle*). What you would like to see is your applications spending more time on user time than system time. Target percentages should be total CPU (*usr* plus *sys*)-80%, *wio*–10%, and *idle*–10%.

CPU overload can be caused by running too many programs or programs that use the system inefficiently. Overhead like too much context switching, too many interrupts, or too many running processes, can also use excessive CPU time. You need to investigate the causes whenever you see this. The *%wio* field in Figure 14 indicates the percentage of time spent in waiting for blocked I/O to complete. If this value reports a significant portion, you need to investigate your I/O performance. If your system is heavily loaded and *%idle* is high, you may have memory or contention problems.

```
% sar -u 5 10

dvlseq dvlseq 3.2.0 V1.4.0 i386     01/13/93

15:23:27       %usr     %sys     %wio    %idle
15:23:32          7       16       76        1
15:23:37         10       15       70        5
15:23:42          9       14       74        4
15:23:47          9       13       71        6
15:23:52          9       12       73        6
15:23:57         13       13       69        4
15:24:02          9       13       71        8
15:24:07         10       16       69        4
15:24:12          6       14       80        0
15:24:18         11       16       73        0

Average           9       14       73        4
```

Figure 14: Sample CPU Activity Display

In multi-processor environments, performance of a system may also be improved by distributing the workload evenly across CPUs. It is not efficient to leave any CPU idle or to saturate any particular CPU. See Tip 48 and your system administration manual for more information.

Tip No. 44: Keep All Oracle Users/Processes at Same Priority

Oracle is designed to operate with all users and background processes operating at the same priority level. Changing priorities may cause an unknown effect on contention and response times.

For example, if the log writer process (LGWR) gets a low priority, it won't be executed frequently enough and eventually LGWR will have a bottleneck. On the other hand, if LGWR has a high priority, user processes can suffer poor response time. Thus, for most UNIX systems, such Oracle background processes need to be scheduled evenly to avoid any bottleneck.

For some platforms, this standard advice can be relaxed. For example, Sun advises using real-time priorities for the Oracle background processes. For some benchmarks, increasing the priority of DBWR improved performance on a Pyramid system. However, most casual modification in process priorities will result in a reduction of overall throughput, so changes should be made with great caution.

Tip No. 45: Use just enough Query Servers

When using the parallel query option, queries are split up for processing in parallel by many Query Servers. It is important to use as many Query Servers as needed to use the available CPU power. However, if too many Query Servers are in use, the system can thrash or become CPU bound. When a query is started, it will be executed in parallel if there are enough query servers available to run it. Simple scans use one Query Server for each degree of parallelism requested. However, for sort-merge (SMJ) or nested loop joins, each query needs two Query Servers for the degree of parallelism requested. If there are not enough Query Servers available to process every piece of a query when it is started, the query will be executed sequentially. For example, if a SMJ query uses a table with a `DEGREE` of 4, at least 8 Query Servers must be available in the system for parallel processing. (Available Query Servers are Query Servers that are idle, or that can be started up.)

The degree of parallelism can be specified for each query and table. The maximum number of Query Servers is specified for each database instance. The values below are starting points. Tip 59 describes how to measure parallel query performance and adjust the parameters accordingly.

- Database Instance: A Query Server will use from 10% to 50% of the power of a single processor when active. To specify the maximum number of Query Servers for an instance, set the *initdb<n>.ora* parameter `PARALLEL_MAX_SERVERS` to a value that allows from 1 to 8 Query Servers per processor in each Oracle instance. (The proper value is very application and system dependent.)

- Tables: Use the `CREATE` (or `ALTER`) `TABLE` (or `CLUSTER`) `PARALLEL` command to specify the `DEGREE` of parallelism for a single instance, and `INSTANCES` to specify the number of instances available when the Parallel Server option is used. The product of `DEGREE` times `INSTANCES` should be equal or less than the number of disk drives the table is striped across (Tip 27). `DEGREE` must be less than one half `(or less than)` `PARALLEL_MAX_SERVERS` for a uni-processor or SMP system.
- Queries: Use the `PARALLEL` and `NOPARALLEL` hints to reduce the parallelism that could be obtained for the tables in the query. This should be done as part of a load balancing or for lower priority queries that use tables designed for highly parallel, high priority queries.

  While it is possible to specify a higher degree of parallelism in a query than is specified for a table, this capability should be used with caution. If a new query requests more Query Servers than the system can provide, that query will be executed sequentially. As a result, a hint requesting a highly parallel query may actually execute more slowly than one using the parallel degree assigned to the table.

Tip No. 46: Control Scheduling

By default, the typical UNIX kernel uses a time-slice scheduler with a simple aging algorithm. Some platforms provide real-time schedulers, which may improve performance, while other platforms provide techniques to "tune" the scheduler which may improve the performance of a particular Oracle installation. This kind of tuning works best for OLTP applications with many short transactions. Some vendors who provide control of scheduling are:

- SVR4 -- Allows the use of *priocntl* to schedule real-time jobs.
- SVR4 -- Has the *pbind* utility to associate a job with a CPU.
- Data General -- DGUX has *NOILEVEL* and *NOLANGUISHING* kernel parameters.
- IBM -- AIX has the *schedtune* utility to adjust the time-slice.
- SCO -- MP supports the *LOADBALANCE,PREEMPTIVE*, and *DOPRICAL* kernel parameters.

Tip No. 47: Reorganize Usage Patterns

If the system is overused during peak periods then look for ways to redistribute the loads to off-peak times. For example, batch processes can be pushed off a few hours or even done overnight. Backups are often made at night. You can also move entire applications to another system or reconfigure your system to make use of the network.

Tip No. 48: Use Processor Affinity/Binding on Multi-processor Systems

Another area you can tune to improve the performance in multi-processor environments is to use processor affinity/binding if it is available on your system. The processor binding prevents a process from migrating from one CPU to another. If the process doesn't migrate, the information in the cache of the CPU where the process originates can be used. Therefore, this tip depends on the cache policy and the architecture of your system. You can bind one of the server processes, a shadow process, to make use of cache since it is always active and let background processes flow between CPUs. Explore the possibility of using processor binding if your system supports it since there are many variations in binding implementation. Some platforms do process binding automatically. Processor binding is a system dependent issue; refer to your system administration manual.

Depending on your initial configuration, your system resource limitations, and your overall system load, the performance benefit that you might see is roughly 0 to 10%.

Tip No. 49: Use a Client/Server Configuration

If your system is CPU-bound, you may want to move applications to a separate system to off-load the CPU. For example, you can off-load foreground processes such as Oracle Report Writer and Oracle Forms to a client machine to free CPU cycles on the database server machine.

Tip No. 50: Use Post-Wait Driver

Oracle processes use semaphores to coordinate access to shared resources. If a shared resource is locked, a process will suspend and wait for that resource to become available. When the resource is available, the process will be posted by having its semaphore incremented. The Oracle Server uses one semaphore per Oracle process. Since semaphores are relatively slow and expensive, any improvement in the posting and waiting for the request and advice can make a significant performance improvement.

One way to improve the function done by semaphores is to use a post-wait driver instead, if it is available on your system. It is a much faster and less expensive synchronization mechanism than a semaphore. Check its availability in your *Installation Guide*. The use of post-wait drivers varies depending on the system.

Depending on your initial configuration, your system resource limitations, and your overall system load, the performance benefit that you might see is roughly 0 to 10%.

Tip No. 51: Use Single-Task for Large Exports/Imports

Linking Oracle executables as single task usually has some danger since a user process can directly access the entire SGA. Furthermore, running single-task requires more memory, since the Oracle executable's text is no longer shared between the front-end and the background processes. However, its advantage is a much faster process (for a system that is not memory-bound). Thus, if you need to transfer large amounts of data between the user and Oracle (such as export/import), it is very efficient to use single-task. To make the single-task import (*impst*) and export (*expst*) executables, use the oracle.mk program which can be found in the $ORACLE_HOME/rdbms/lib directory.

Depending on your initial configuration, your system resource limitations, and your overall system load, the performance benefit that you might see is roughly 0 to 15%.

Tip No. 52: Expand Your System

If none of the above is successful, consider buying more CPUs or cache, or upgrading to larger or faster CPUs.

XI. Step 7: Tuning Resource Contention

This section describes a number of techniques for gathering information on resource contention. In particular, tables of "latches" and "events" are examined. Before using these tips, however, you should be asking yourself if you suspect a contention problem. Usually, a contention problem is suspected when poor database performance is observed, yet the normal UNIX measures of contention, CPU and disk activity do not indicate a problem. Tips 53 though 56 describe the use of Oracle internal tables to find contention, gather more detail on the contention, and finally isolate the SQL which causes the contention. After that, we examine additional database contention issues (Tips 57 - 60) and reducing memory contention through UNIX kernel size reduction (Tip 61).

Tip No. 53: Use v$ Tables to Isolate Contention

Many consultants in Oracle's *Open System's Performance Group* use the *v$system_event* table to get quick insight on database activity. The effective use of this table requires a firm understanding of Oracle operation. Many of the parameters which these tables measure can be found in the *Server Concepts* manual and the *Server Reference* manual.

The statistics in *v$system_event* tell us how the Oracle RDBMS kernel is using its time, and we can use this information to infer potential problem areas in the database. The table can be examined using:

```
sqldba> select * from v$system_event order by
time_waited;
```

The output of this command appears in Figure 15.

The number of rows in this table will change dynamically. If there is no information to report on an event, then the event will not appear in the table. A well tuned database will experience waits, and the presence or absence of an event in this table does not necessarily indicate a problem. In fact, it is normal to see events such as *client message, pmon timer, smon timer, rdbms ipc message*, and *rdbms ipc reply*, among others.

Typically, after examining *v$system_event*, more detail on the events is desired. Since *v$system_event* is a cumulative table, it will be useful to look at a table which measures the same events as they occur. This is done with the *v$session_wait* table as follows:

```
sqldba> select sid, event, p1text, p1, p2text, p2 from
v$session_wait;
```

Database Administrator's often run this query iteratively, since it gives a "snapshot" of events. It can be very insightful to see how an event's frequency changes with load on the database, giving insight into both Oracle operation and the

```
EVENT                          TOTAL_WAITS  TOTAL_TIMEOU TIME_WAITED  AVERAGE_WAIT
------------------------------ ------------ ------------ ------------ ------------
buffer busy waits                         1            0            0            0
instance state change                     2            0            0            0
control file sequential read            192            0            1 .00520833333
latch free                                8            5            5         .625
free buffer waits                         3            0            5 1.6666666667
log file sequential read                 11            0            6 .54545454545
log file single write                    12            0           23 1.9166666667
db file single write                     42            0           52 1.2380952381
log file space/switch                     3            0           66           22
control file parallel write             102            0          153          1.5
rdbms ipc reply                           7            0          159 22.714285714
write complete waits                     27            0          190  7.037037037
log file sync                           167            0          365 2.1856287425
log file parallel write                 393            0          916 2.3307888041
db file sequential read                7028            0         1731 .24630051224
db file parallel write                  231            0         2207 9.5541125541
db file scattered read                22041            0        30110 1.3660904678
smon timer                                7            3        95687 13669.571429
client message                         5091            0       102196 20.073855824
pmon timer                              385          385       115490 299.97402597
rdbms ipc message                      1047          768       233032 222.57115568
parallel query dequeue wait           13466        13476      2693229 200.00215357
22 rows selected.
```

Figure 15: v$system_event Output (time in hundredths of seconds)

```
SID  EVENT                    P1TEXT       P1         P2TEXT     P2
 1   pmon timer                            0                     0
 2   rdbms ipc message        timeout      300                   0
 3   rdbms ipc message        timeout      133                   0
 5   rdbms ipc message        timeout      180000                0
10   enqueue                  name|mode    1.42E+09   id1        65551
 6   buffer busy waits        file #       7          block#     792
 8   db  file  sequential     file#        7          block#     792
     read
 9   db  file  sequential     file#        7          block#     1040
     read
11   db file scattered read   file#        7          block#     1040
 4   smon timer               sleep        300        failed     0
                              time
14   client message           two-task?    1          driver id  4616808
10   latch free               address      8.05E08    number     8
```

Figure 16: v$session_wait Output

nature of the SQL which is being executed. For example, repeated execution may result in an event appearing for a length of time, and that event may be a result of contention caused by SQL executed during that period. See Figure 16 for sample output of *v$session_wait*.

Some possible interpretations of the events of Figure 15 and Figure 16 are shown in Figure 17.

Event	Meaning	Some possible causes	Actions
buffer busy waits	This indicates contention for a buffer in the SGA. Typically, another process has the buffer locked.	Segment header contention. Index or data block contention. Free list or rollback block contention.	See Tip 54.
enqueue	An enqueue is used to keep track of processes which are waiting for a locked resource, including sequences.	Too many DDL locks. Too many DML locks. Excessive use of Sequences.	See Tip 55 and Tip 9
free buffer waits	A process is waiting for a free buffer in the SGA. This buffer may be of any type and for any use.	DBWR can't write enough buffers. Checkpoints poorly adjusted.	See Tip 31
db file scattered read	This measures hard disk read operations which read multiple blocks in a single operation.	SQL which does full table scans.	See Tip 55 and Tip 9
db file parallel write	This measures hard disk write operations which issue multiple write calls and wait for completion	The buffer cache may not be large enough to be effective.	See Tip 38
latch free	This is a "generic" latching statistic, which monitors all latches which are allocated and deallocated.	Many possible causes.	See Tip 56

Figure 17: Interpreting v$session_wait & v$system_event

Tip No. 54: Isolate the Segment causing contention

If you decide that *buffer busy waits* is a concern, you will notice that Figure 16 displays both the block number and file number where the contention is occurring. These can be used to determine the type of contention as follows:

```
sqldba> select segment_name, segment_type, block_id,
blocks from dba_extents where file_id=7 and (792
between block_id and block_id+blocks-1);
```

Sample output follows:

```
SEGMENT_NAME     SEGMENT_TYPE     BLOCK_ID BLOCKS
---------------  ---------------  -------- --------
COT1             TABLE                 752       50
1 row selected.
```

This indicates that the contention is occurring in a table segment, as opposed to an index, cluster, or rollback segment. Since we have the file# and block#, it is possible to get some additional information from the x$bh table as follows:

```
sqldba> select class from x$bh where dbafile=7 and
dbablk=792;
```

This provides the *class* of the block, which can be interpreted using the following table:

Class	**Block Type**
0	System rollback segment
1	Data block
2	Sort block
3	Save Undo block
4	Segment header block
5	Save Undo segment header block
6	Free List block
7 + (n*2)	Undo segment header block
7 + ((n*2)+1)	Undo segment block

This allows some additional classification of the contention. Figure 18 gives some examples of possible causes of contention.

Segment Type	**Class**	**Causes/Actions**
Table	Data block	Application locking Problem? Check the locking sequence of your tables.
Table	Segment header block	Free list contention? Create more free lists using *create table*. See Tip 63.
Index	Data block	Sequences causing contention in a leaf of the index? See Tip 62. Parallel loading with an index? Drop the index before loading, and create index in parallel later.
Index	Segment header block	Free list contention? Create more free lists using *create table*.

Figure 18: Possible causes of contention

Tip No. 55: Isolating the SQL associated with an *event*

If, for example, we observed *buffer busy wait* contention in Figure 15 or 16, it is possible to determine the SQL which is causing the contention using the *SID* field from the output of *v$session_wait.*

This is done by examining the *SQL cache* in the SGA which stores currently running SQL.

```
sqldba> select sql_text from v$sqltext, v$session
where  sid=6  and  v$sqltext.address=v$ses-
sion.sql_address;
```

Output from this command displays the SQL which caused the contention:

```
SQL_TEXT
----------------------
select f1,f2,f3,f4 from c0t1
1 row selected.
```

Of course, this same information can be obtained using a single select and a join with the *sid* to ensure that the cached *sql_text* is not out of date.

Tip No. 56: *Latch Free* contention

If, while examining the output from *v$system_event*, the value of *latch free* seems to be causing contention, then the output from *v$session_wait* can be used to determine which latches are causing contention. The latch number is given in the *P2* field, and can be identified with a query of the form:

```
sqldba>select latch#, name from v$latch where
latch#=8;
```

An example of output is:

```
LATCH#   NAME
-------- -------------------------
       8 cache buffers chains
1 row selected.
```

The *cache buffer chains* latch often experiences contention, as do the *cache buffer lru chain* latch and the *cache buffer handles* latch. These latches typically indicate that tuning the number of SGA buffers is required. See Tip 19

Tip No. 57: Reduce Rollback Segment Contention

The data files have segments which are allocated for rollback information. These segments hold copies of data which are being changed by running transactions. This means every transaction that changes data will generate rollback information. This rollback information is used not just for transaction rollback, but also for "point in time" read consistency and database recovery. Since the database blocks that make up rollback segments are accessed frequently, rollback segments may be subject to contention.

Use the following SQL statement to determine how often requests for space in a rollback segment cause delays. The hit rate (`HITS`) should be more than 95%. Figure 19, below shows output results from this SQL statement.

```
SELECT name, gets, waits, ((gets-waits)*100)/gets hits
    FROM v$rollstat s, v$rollname n
    WHERE s.usn = n.usn;
```

If too few rollback segments exist, or if they are too small, or if users are not assigned to rollback segments properly, then a rollback segment may become a bottleneck. The simple solution to rollback contention is to add more rollback segments (Tip 6).

It is also a good idea to assign users who run large transaction to large rollback segments so that they don't run out of rollback space. A common symptom of too little rollback space is the error message "snapshot too old."

Sample output (from an inactive system)

```
NAME     GETS    WAITS    HITS
------   ----    ----     ------
SYSTEM   133      0        100
R01      12       0        100
R02      20       0        100
R03      33       0        100
R04      11       0        100
5 rows selected.
```

Figure 19: Rollback Segment Contention SQL Output

Tip No. 58: Reduce Redo Log Buffer Latch Contention

The *redo log buffer* is a circular buffer in the SGA that holds information about changes made to the database. Access to the buffer is regulated by latches. Since heavy access to the redo log buffer can result in contention for the redo log buffer latches, it is a good idea to examine the activity of the redo log buffer latches through *Server Manager's* latch display or *sqldba* as follows:

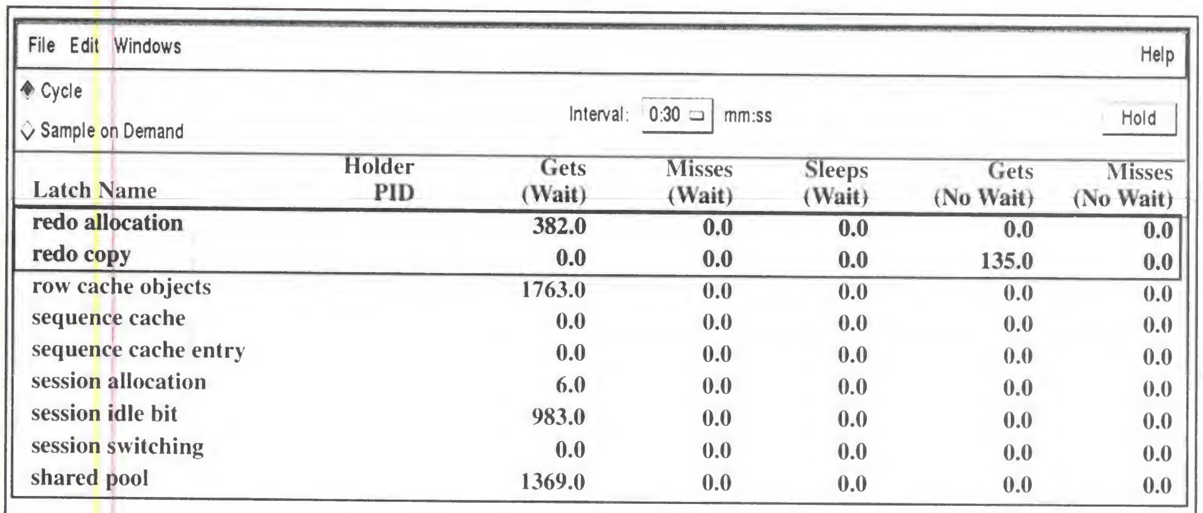
```
SQLDBA> monitor latch
```

Figure 20 shows a sample output for the equivalent display from *Server Manager*. Examine the statistics for the *redo allocation* latch and the *redo copy* latches of type *Wait*:

Gets	Number of successful requests for the latch
Misses	Number of times latch was in use when requested
Sleeps	Number of times latch request timed out

If the ratio of *Misses* to *Gets* for a particular latch exceeds a threshold of 10% or 15%, contention for that latch may be affecting performance. Each *Sleep* indicates a delay for the process requesting the latch. Since some systems with many CPUs may be able to tolerate more contention without performance reduction, refer to your *Installation & Configuration Guide* for more specific information on your platform.

To reduce contention for the *redo allocation* latch, minimize the time that any single process holds the latch by decreasing the value of the `init<n>.ora` parameter LOG_SMALL_ENTRY_MAX_SIZE. One way to reduce contention for *redo copy* latches in multi-processor environments is to add more latches by increasing the value of LOG_SIMULTANEOUS_COPIES in `init<n>.ora`. It can help to have up to twice as many redo copy latches as CPUs available to your Oracle instance. You can also reduce contention for redo copy latches by reducing the time each process holds a latch through forcing the Oracle user process to "pre-build" redo entries before obtaining a redo copy latch. Additional information is available in the references described in the Bibliography.



Latch Name	Holder PID	Gets (Wait)	Misses (Wait)	Sleeps (Wait)	Gets (No Wait)	Misses (No Wait)
redo allocation		382.0	0.0	0.0	0.0	0.0
redo copy		0.0	0.0	0.0	135.0	0.0
row cache objects		1763.0	0.0	0.0	0.0	0.0
sequence cache		0.0	0.0	0.0	0.0	0.0
sequence cache entry		0.0	0.0	0.0	0.0	0.0
session allocation		6.0	0.0	0.0	0.0	0.0
session idle bit		983.0	0.0	0.0	0.0	0.0
session switching		0.0	0.0	0.0	0.0	0.0
shared pool		1369.0	0.0	0.0	0.0	0.0

Figure 20: *Server Manager* Latch Display

Tip No. 59: Reduce Parallel Query Contention

Parallel queries should be tuned both to prevent using too much CPU and also to prevent exhausting the supply of available Query Servers. The `V$PQ_SYSSTAT` view can be used to learn how many Query Servers are in use as shown below:

```
SELECT * FROM V$PQ_SYSSTAT
    WHERE statistic = "Servers Busy":

STATISTIC                VALUE
---------------------    -----------
Servers Busy             70
```

If this number reaches the value set for `PARALLEL_MAX_SERVERS`, it means that at least some parallel queries are probably being processed sequentially. At the same time, run *sar -u* to observe CPU loading. Observe these measurements over a significant period of time. The table below summarizes tuning actions based on the ratio of Servers Busy to `PARALLEL_MAX_SERVERS` compared to CPU utilization.

Busy/Max Servers	**CPU use heavy (95 - 100%)**	**CPU use OK (60 - 80%)**	**CPU use light (0 - 30%)**
1.0 often	• Aggressively decrease parallelism in tables and queries • Tune system	• Decrease parallelism in tables and queries.	• Increase MAX servers. • Tune the system.
1.0 rarely	• Identify queries when maxed, tune. • Tune the system.	• Increase MAX Servers - watch • Decrease parallelism in tables and queries.	• Increase MAX servers.
.3 - .7	• If Query Servers using >40% CPU, decrease MAX servers • Tune the system.	• Tuned	• Increase parallelism in tables and queries. • Increase MAX servers.
0 - .2	• Tune the system. • Consider adding processors	• Consider lowering MAX servers.	• Increase parallelism in tables and queries.

Tip No. 60: Tune Spin Count on Multi-processor Systems

In multi-processor environments, you can improve the performance by tuning SPIN_COUNT, which is a parameter in `initdb<n>.ora`. The spin count is basically a loop counter such that a process keeps on requesting a latch until it gets one. If the process loops up to the spin count (i.e., fails to acquire the latch), it goes to sleep and then it tries the loop again to get the latch after it wakes up. Since a latch is a low level lock, a process doesn't hold it too long. Therefore, it is cheaper to waste CPU time by spinning a process than making a process sleep.

You can check the contention level of the latch by monitoring the *miss rate* and the *sleep rate* from `utlbstat/utlestat`. The goal is to reduce the sleep rate by tuning the spin count. If the contention level is high, you can increase the spin count to make processes spin more before acquiring the latches. However, since increasing the spin count makes CPU usage go up, the system throughput may go down at some point. Thus, it is important to find the optimal point by tuning spin count.

Tip No. 61: Tune UNIX Kernel Parameters

One suggestion to improve performance is to keep the UNIX kernel as small as possible. This helps because the UNIX kernel typically preallocates physical RAM, which makes less memory available for other processes, such as Oracle.

Traditionally, parameters such as NBUF, NFILE, and NOFILES were used to adjust kernel size. In recent years, however, most UNIX implementations dynamically adjust those parameters at run time, even though those parameters are still seen in the UNIX configuration file.

It is often more useful to look for device drivers which are configured into UNIX by default, but aren't used in your installation. In particular, memory mapped video drivers, networking drivers, and disk drivers can often be uninstalled, yielding more memory for use by other processes.

Of course, remember to make a backup copy of your UNIX kernel to boot from, just in case you delete a driver that you really need. See your hardware vendor's documentation for additional details.

XII. Step 7: Tuning Resource Contention for the Parallel Server Option

The Oracle Parallel Server option(OPS) allows many independent UNIX processors with separate Oracle Instances to operate on a single database stored on collection of shared disks. Database concurrency is maintained across the processors using a Distributed Lock Manager (DLM). Normally the actions of the DLM are transparent to the users of the database. However, managing resources using the DLM is less efficient than using the shared memory model within a single database instance. The tips below describe tuning techniques designed to minimize the chances that the DLM will become a bottleneck.

Tip No. 62: Avoid Index Contention

Index Tables are often extensively used, and may be a source of contention in your database. This problem often arises when a sequence generator is being used to create primary keys for database records. The sequence numbers are typically consecutive, and, when used as keys to add data, cause entries in the same index blocks. This can result in contention for the index blocks. This problem is solved by modifying the sequence number in some way before adding a record: This may be done by pre-pending values to the sequence.Values are selected to distribute indexes to different blocks. (This can also be an issue for the Oracle Server running on an SMP, but is more likely to become a bottleneck on the OPS.).

Tip No. 63: Avoid Free List Contention

Blocks which are available for insert operations are kept on a list. By default, this list is kept in the table header. That means that an insert intensive application will experience contention for the table header block. This problem is solved by creating multiple free lists and multiple free list groups. Free list headers are kept in different blocks. Freelists are allocated during the creation of tables or indexes.

Tip No. 64: Avoid Lock Contention

If there is too much lock contention, then an application will not scale well. Lock contention can be measured by fields from the *v$sysstat* table (CLASS=32)

$$\text{Lockconversionratio} = \frac{\text{Consistentgets} - \text{Asynclockconverts}}{\text{Consistentgets}}$$

This ratio should be 95% or higher in order for the application to scale well. If there is too much lock contention then the application must be re-evaluated and perhaps re-designed to work with the Parallel Server option.

Although the application being executed by the Oracle RDBMS is the most common source of lock contention, sometimes insufficient locks have been allocated, or they were poorly allocated. Remember that a Transaction Processing application will need many more locks that a Decision Support application, and allocate locks appropriately with the `init<db>.ora` parameters.

Tip No. 65: Localize Disk I/O

Keep rollback segments for an instance on disks which are connected to that same node. Also, keep redo logs associated with an instance on disks connected to that node. This should be part of the overall strategy of partitioning data so that each node tends to use data without contention. Remember that both redo and rollback should be on shared disk, so that they are available to other nodes in the event of an instance failure.

Tip No. 66: Monitor Contention

Many statistics are kept which might indicate Parallel Server option contention. Some tables to examine are: *v$session_wait, v$session_event, v$system_event, v$sql_area, v$cache*, and *v$ping*. See Tip 53 through Tip 58

In general, lock conversions are the most important parameter. Lock conversions imply disk I/O and delays while the lock is acquired and converted. Proper application partitioning is the only sure way to avoid lock conversions.

Appendix 1: Oracle Optimal Flexible Architecture (OFA) Rules

Optimal Flexible Architecture Overview

The *OFA Standard* defines installation guidelines that will give you high performing, reliable databases requiring less work to maintain. OFA is specified as a set of 13 *Requirements* and 11 **OFA** rules. This appendix describes OFA and illustrates OFA for UNIX systems. For more detailed information about OFA, please get a copy of *The OFA Standard: Oracle7 for Open Systems* by Cary V. Millsap, Part Number A19308-2, or contact Oracle Services.

When installing the Oracle Server, tools, applications, and databases many decisions about where to place files, how to allocate data across disks, etc. are possible. Oracle allows great flexibility for these installation decisions. Following the OFA guidelines will result in a mapping of all of the Oracle binary, control, administration, trace, and data files appropriately onto the UNIX file system. This appendix includes OFA compliant examples, including examples using symbolic links, Network File System (NFS) partitions, and datafiles on raw devices (UNIX character special files).

New installations and small to mid-sized databases

If you are doing a new installation or building a new database, you can use the installer to build an OFA compliant database. Figure 1 shows such a newly installed OFA compliant installation. You will have to supply a database name (**db_name**). Use a descriptive 8 character or shorter name.

Installing an OFA compliant database and Oracle software products with the version of the Installer shipped with the Oracle7 Server Release 7.2 requires entering the path names for the data files rather than accepting the suggested defaults. After the install is completed, you will need to build the *admin* directory structure and change the dump file destinations in $ORACLE_HOME/dbs/config**<db_name>**.ora to point to the appropriate directories. You may also want to move the data base creation scripts and parameter files to the ../admin/**<db_name>**/(pfile or create) directories. If you do move the parameter files out of $ORACLE_HOME/dbs, you should create a symbolic link in dbs named init**<db_name>**.ora to ../admin/**<db_name>**/pfile/init**<db_name>**.ora. Failing to do this will require using the PFILE modifier during database startup in SQL*DBA and/or Server Manager.

Following this initial installation, create additional mount points (**u02**, etc.) as described by **OFA 1**. Then add any additional data files as described by **OFA 6**. (See your *Installation & Configuration Guide* or the *Server Administrator's Guide*, Ch. 2 for the procedures to follow.)

The version of the installer planned to be shipped with later versions of the Oracle7 Server will include additional support for OFA. For larger databases, or databases built on raw devices, it will still be better to create the database from a SQL script run after the Oracle software products are installed.

The rest of this appendix describes OFA in detail. You can use it to learn how to deal with the more complex database questions that arise when very large databases are used, new software is installed, applications exceed the size of a single disk, etc.

I. File System Configuration

UNIX systems usually have one or more physical disks. Each disk drive is formatted into one or more fixed pieces called *partitions, sections* (HP), or *slices* (Sun). These disk slices are then used as the basic building blocks for the UNIX file system and Oracle data files. OFA needs to ensure that the file system (and raw devices) is built in a manageable way across the disk drives in the system. Ideally, determining what sized slices to use and then using these slices to build up the UNIX file system and/or Oracle data files should be planned and completed before installing Oracle software.

Logical Volume Managers

On systems with a Logical Volume Manager (LVM), such as HP, Sun, DG, IBM, Pyramid, and UNIXware, a *Logical Volume* can be built up from slices on several physical disks. These logical volumes can be used instead of slices to build up the file system and datafiles. With an LVM, a logical volume can be much larger than a single physical disk. In addition, Logical Volumes can stripe the data from a single(large) file across many disks, potentially increasing performance (Tip 27). Oracle recommends using an LVM, if available.

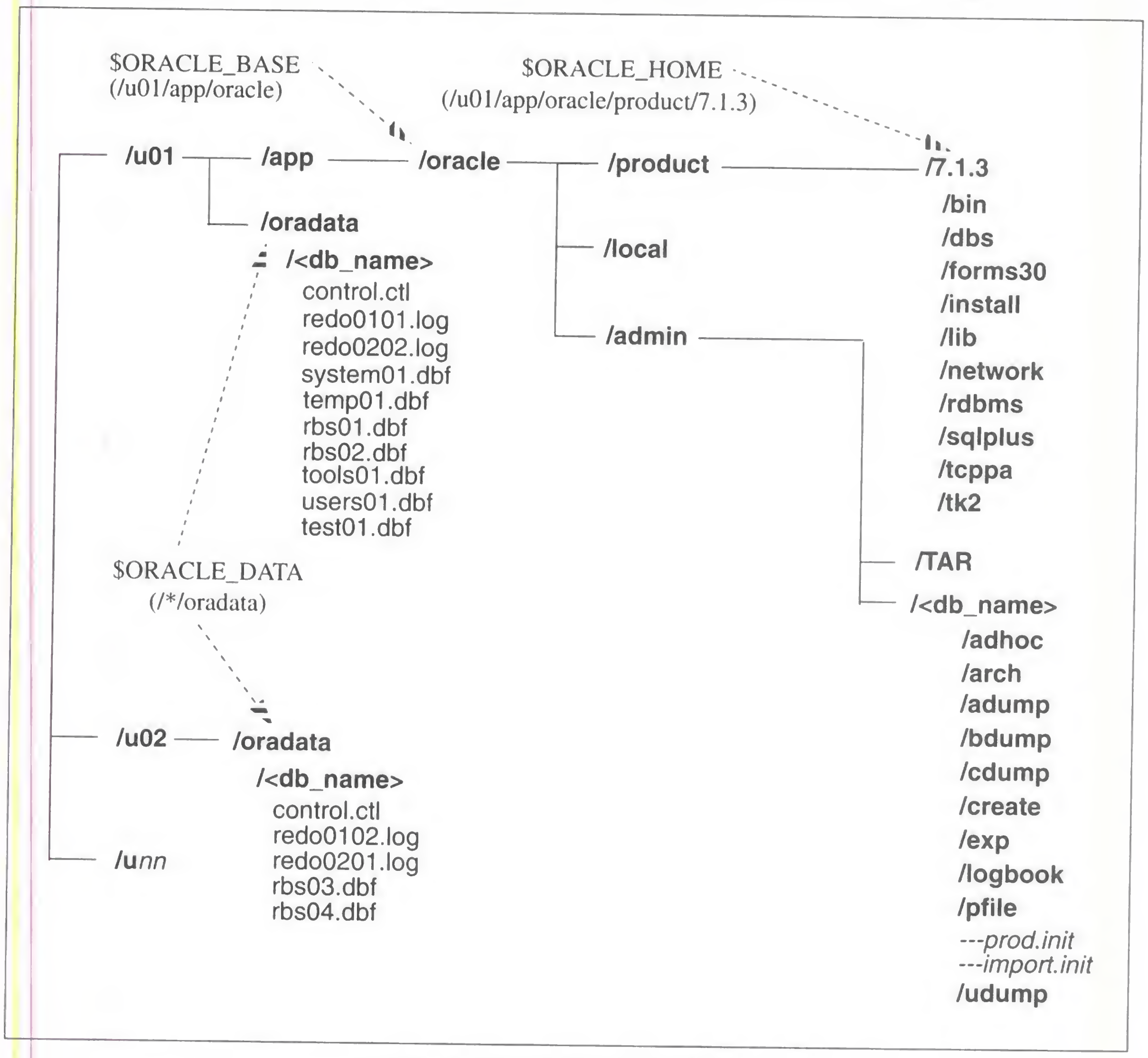


Figure 1: Default Oracle Installation

Mount Points

A *mount point* is a directory name identifying where the file subsystem for a single disk slice (or logical volume) will be linked into an existing UNIX file system. You will have to choose mount points for slices containing all the software and data files. Careful selection of mount point names can make UNIX easier to administer. In selecting names, the configuration planner must find an appropriate balance among these requirements:

Requirement 1. The file system must be organized to allow easy administration of growth: Adding data into existing databases, adding users, creating databases, and adding hardware

Requirement 2. .It must be possible to distribute I/O load across sufficiently many disk drives to prevent a performance bottleneck.

Requirement 3. It may be necessary to minimize hardware cost.

Requirement 4. It may be necessary to isolate the impact of drive failure across as few applications as possible.

Using an LVM to build a single logical volume across every disk satisfies requirement 2 with minimal effort. Unfortunately it does not allow adding a new disk easily, and a single disk failure would prevent every application from running until repair and recovery was completed, violating requirements 1 and 4. In general, meeting requirement 3 will usually result in some compromise to requirements 2 and 4. For an LVM, the logical volumes can be striped across only some of the disks, or be built as mirrored logical volumes, meeting all four requirements.

One way to balance these requirements is to name every one of your UNIX mount points so it is easy to find where each disk slice in the file system. The following rule accomplishes this goal.

OFA 1

Name all mount points that will hold site specific data to match the pattern */pm* where *p* is a string constant and *m* is a unique fixed-length key used to distinguish each mount point. Examples: **/u01** and **/u02** or **/disk01**, **/disk02**, etc.

VLDB Mount Points

Some sites have a single Oracle database large enough to require many disk drives of storage. These sites are able to satisfy *Requirement 2* and still dedicate entire disk drives (or logical volumes made up of entire disk drives) to a single Oracle database application. In this case a different mount point naming strategy can be considered.

OFA 11

1. If you can guarantee that each disk drive will contain database files from exactly one application, and...
2. You have enough drives for each database to ensure that there will be no I/O bottleneck.

If 1 and 2 are correct, then name mount points matching the pattern */qdm* where *q* is a string denoting that Oracle data is to be stored there, *d* is the value of the **db_name** init**<db_name>**.ora or config**<db_name>**.ora parameter for the database, and *m* is a unique fixed-length key that distinguishes one mount point for a given database from another. Example: **/ora/intl01** implies that data from the Oracle database **intl** is stored on this drive.

Login Home Directories

Requirement 5. It must be possible to distribute across two or more disk drives both (a) the collection of home directories and (b) the contents of an individual home directory.

OFA 2

Name home directories matching the pattern */pm/h/u*, where *pm* is a mount point name, *h* is selected from a small set of standard directory names, and *u* is the name of the owner of the directory.

Example: the Oracle Server software owner home directory might be **/u01/app/oracle**, and the Oracle Applications software owner home directory might be **/u01/app/applmgr**. Placing all home directories at the same level in the UNIX file system means that we can put home directories on different mount points, yet still be able to refer to the collection of login homes with a single pattern. The pattern **/*/app/*** can be used to refer to all applications owner directories on the system, meeting *Requirement 5*(a).

Using Symbolic Links

In the example above, if both Oracle Financials and Manufacturing share the same software owner directory: **/u02/app/applmgr**, more than a gigabyte of disk space would be needed. *Requirement 5*(b) can be met using symbolic links to make directories appear in a single subtree, even though they physically reside on different mount points.

Figure 2 shows the symbolic link required to enable the Oracle General Ledger software to live on a separate mount point from the other applications software, yet appear to live in **/u02/app/appmgr**. All applmgr files are still identifiable as residents of subtrees whose names match the pattern **/*/app/applmgr**

```
$ ln -s /u03/app/applmgr/gl /u02/app/applmrg/gl
$ ls -l /u02/app/applmgr
-rw-r--r--  1 applmgr      1119 Jul   5 01:16 AOL.env
drwxrwxr-x  1 applmgr      2048 Jul   5 01:16 alr
drwxrwxr-x  1 applmgr      2048 Jul   5 01:16 fnd
lrwxrwxrwx  1 applmgr         5 Jul   5 01:16 gl -> /u03/app/applmgr/gl
...
$ ls -l /uo3/app/appmgr
drwxrwxr-x  1 applmgr      2048 Jul   5 01:16 gl
```

Figure 2: A symbolic link used to distribute a sub-directory across disks

Moving directories

Requirement 6. It must be possible to add or move login home directories without having to revise programs that refer to them.

OFA 3

Refer to explicit path names only in files designed specifically to store them, such as **/etc/passwd** and **/etc/oratab**; refer to group memberships only in **/etc/group**.

II. Oracle Files

Oracle products are made up of many files containing information in different categories. Experience has shown that careful separation of these categories into different parts of the UNIX file structure is essential for safe and simple long-term administration of Oracle applications and data. The separate portions of the file system are then identified using well chosen naming conventions to show where files of each category are located.

The categories of Oracle files include:

- *Product files* consist of Oracle Server software and tools that are supplied on the Oracle Corporation distribution media.
- *Administrative files* are files containing data about the database or instance, including archived redo log files, server process diagnostic output, database creation scripts, on-line exports, instance parameter files, etc.
- *Local software* is software used with Oracle that is written on site or purchased separately from the Oracle distribution software.
- *Database files* consist of control files, redo log files, and data files. Some or all of these files may be stored on raw devices.

All Oracle software and administrative data resides in subtrees of the Oracle Server owner's login home directory. **ORACLE_BASE** is set to the name of this directory.

Requirement 7. Categories of files must be separated into independent UNIX directory subtrees so that files in one category are minimally affected by operations upon files in the other categories.

Product files

Mature production sites normally support two or more versions of a product: the version in production and the version being prepared for production.

Requirement 8. It must be possible to execute multiple versions of applications software simultaneously. Cutover after upgrade must be as simple for the administrator and as transparent for the user as possible.

OFA 4

Store each version of the Oracle Server distribution software in a directory matching the pattern h/product/v, where h is the login home directory of the Oracle software owner, and v represents the version of the software.

Example: **/u01/app/oracle/product/7.1.3** names the start of the directory structure where the Oracle7.1.3 Server files are located. The environment variable **$ORACLE_HOME** is set to this point in the directory structure for the active version of the Oracle Server. (Keep these directories clean. (*Requirement 7*) Adding site-specific data may imperil the next update if the old version directory is deleted.)

Administrative Files

In the normal course of operation, installers store programs that create databases; the Oracle Server itself produces trace files; and administrators save structural records, instance parameters, performance statistics, backups, archives, and general logbook entries on each database. Each database needs a large set of administrative files. The more databases, the more files must be managed.

> *Requirement 9.* Administrative information about one database must be separated from that of others; there must be a reasonable structure for organization and storage of administrative data.

OFA 5

For each database with whose **db_name** is *d*, and where *h* is the Oracle software owner's home directory, store database administration files in the following subdirectories of *h*/**admin**/*d*:

adhoc	ad hoc SQL scripts for a given database
arch	archived redo log files
adump	audit files (Oracle7 audit feature - must clean out periodically) (set audit_file_dest in config<db_name>.ora to point here)
bdump	background process trace files
cdump	core dump files
create	programs used to create the database
exp	database export files
logbook	files recording the status and history of the database
pfile	instance parameter files
udump	user SQL trace files

Some administrative directories, such as arch and exp, are typically too large to store on the disk slice housing the Oracle owner's login home directory. These directories can be connected easily into the administrative subtree with symbolic links similar to the one shown in Figure 2. Using symbolic links is a simple mechanism for storing a file anywhere it needs to be without sacrificing the organization of the file system to physical size constraints.

Local Software

The OFA standard encourages the administrator to add site-specific Oracle software into the system in the **local** subtree of the directory structure described above. In addition, Oracle Consulting will populate this subtree with administrative utilities during an on-site engagement.

File Sharing

Some Oracle customers run substantially the same software on several systems connected together with a high-performance Local Area Network (LAN). Often a portion of the file system is made available over the network using products like NFS (Network File System), DFS (Distributed File System), or AFS (Andrew File System). In these cases, networked partitions can be included in an OFA conforming system as separate mount points. You should use a different */pm* (like **/n01**) to distinguish these partitions. The product, administrative, and local files described in this section and illustrated in Figure 2 can all be part of a network file system and shared. However, see the warnings below about extending this use of networked file systems to database files.

NFS WARNING data files

Do not place any Oracle Database File in any kind of network file system. The concurrency controls on network file systems are not safe when used with data items smaller than a file. The Oracle Server manages data at the block level, much smaller than a file. Even if you think that only one system is using the database files, individual blocks can still be lost or arrive out of order, corrupting your database.

NFS warning other files

Other files may be placed in NFS safely. However, NFS files will typically be slower than local files. NFS is often used to store the binaries, message files, etc. especially on parallel server implementations. However, if the NFS sever fails, every Oracle instance associated with it will typically soon hang or fail. If the parallel server is used for availability, having a single NFS server defeats the purpose. Oracle software is often on NFS during development and prototyping, then moved to regular partitions for mission-critical production use.

Figure 3 opposite shows how the OFA requirements and rules for Mount Points and Oracle software, administrative, and local files come together in a UNIX directory structure. In this case, there are three mount points: **/u01**, **/u02**, and **/u03**. In addition, there are three databases: **intl**, **sab** and **sabt**. Symbolic links are used to present a single, consistent view of the Oracle files independent from the underlying disk drives.

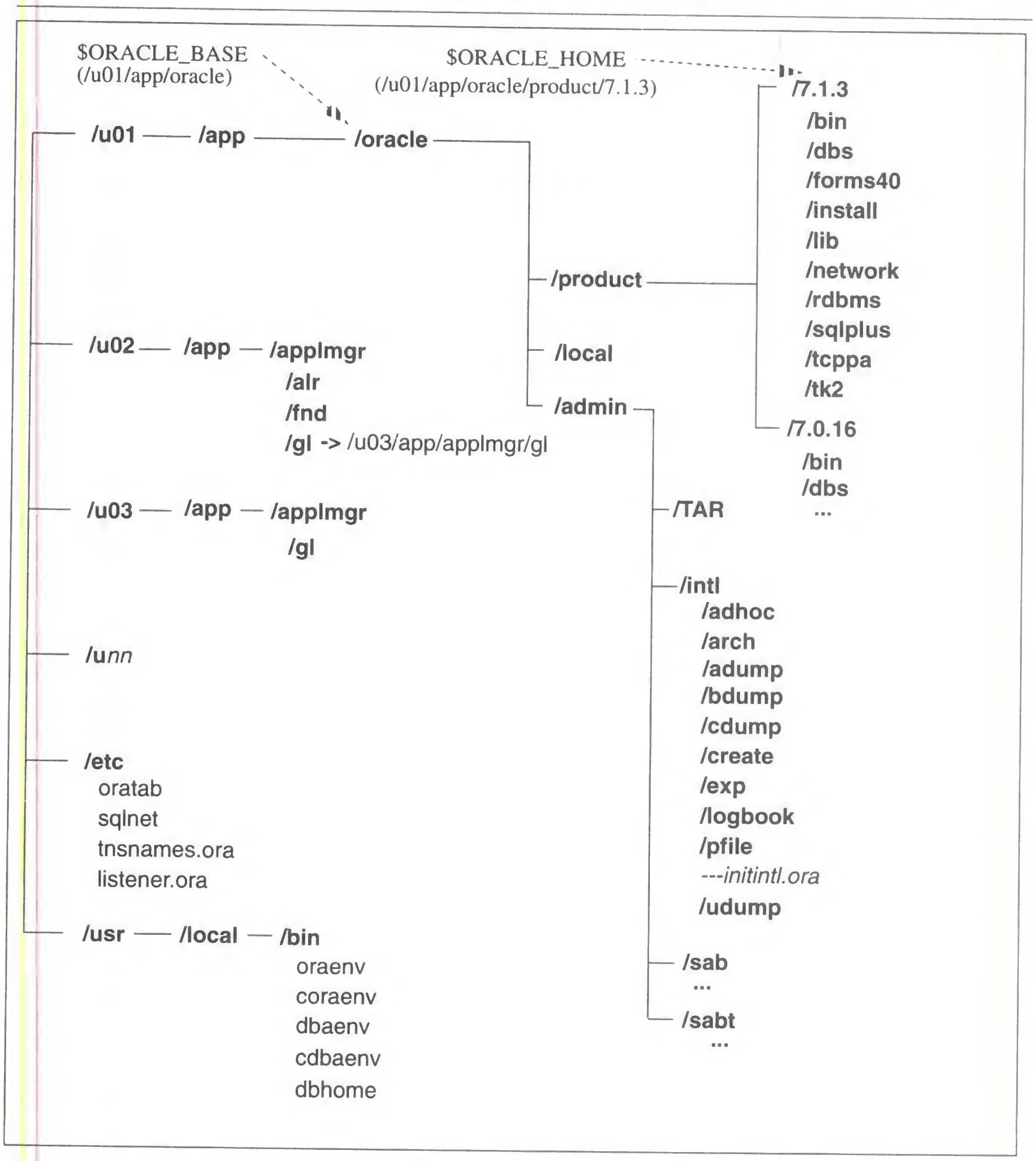


Figure 3: Product and Administrative Files

III. Oracle Database Files

Oracle database files should be separated from the other files on the system for several reasons. Database files have lifetimes that differ from all other files on the system. Database files also require a different backup strategy than the other files. Experience yields the following requirement:

Requirement 10. Database files should be named so that (a) database files are easily distinguishable from all other files; (b) files of one database are easily distinguishable from files of another; (c) control files, redo log files, and data files are identifiable as such; and (d) the association of data file to tablespace is easily identifiable.

OFA 6

Name Oracle database files using the following patterns:

*/pm/q/d/***control.ctl**	control files
*/pm/q/d/***redo***n***.log**	redo log files
*/pm/q/d/tn***.dbf**	data files

where *pm* is a mount point name, *q* is a string separating of Oracle data from all other files, *d* is the **db_name** of the database, *n* is a key that is fixed-length for a given file type, and *t* is an Oracle tablespace name. Never store any file other than a control, redo log, or data file associated with database *d* in */pm/q/d*.

/pm is the mount point name discussed earlier. The *q* layer is normally named **ORACLE** or **oradata** satisfying *Requirement 10* (a). Similarly, naming the *d* layer **db_name** satisfies *Requirement 10* (b). (c) and (d) in *Requirement 10* are met in by including the names for file types and tablespaces directly in the path.

Figure 4, opposite, shows an OFA compliant directory structure for storing I/O balanced database files that are independent of the Oracle software and administrative files. In this example, *q* = **oradata** on a system with three databases and three mount points. Remember to replicate the directory structure on every mount point on the system, even if no database files will be put there initially. In this way, if will be easy to add new datafiles with confidence, even on short notice.

Control files

Oracle control files contain structural information about the database, including relatively static data like the current redo log sequence number. For safety's sake, it is essential that the administrator create at least two control files on two different disks. Having three control file copies ensures that even if one file is lost, there remains a safe duplication. Because control file copies are always stored on different disks, they can have identical basenames.

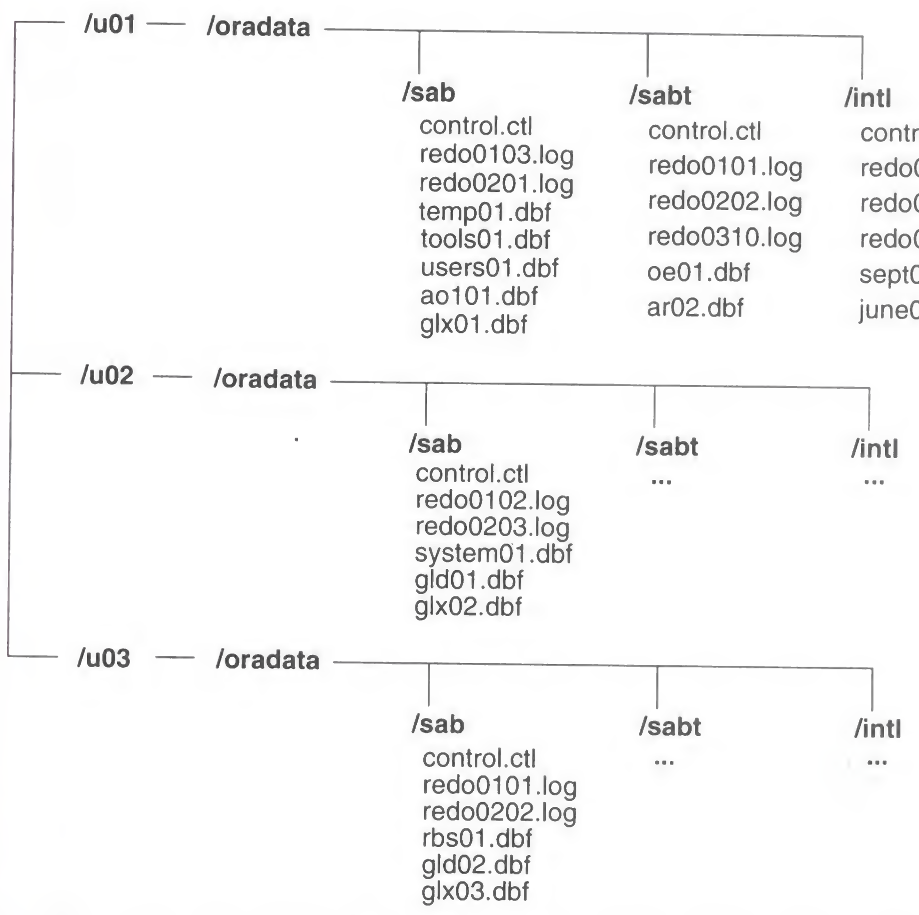


Figure 4: OFA-Compliant Database file structure.

Redo log files

Oracle redo log files record information necessary to roll forward a database in case of CPU or disk failure. Redo log files are written sequentially during transaction processing, and they are read only during archiving and recovery. Since redo log files for a database may all exist on a single drive, the file basename must be distinguished. The distinguishing key is normally a three- or four-character string showing the group and sequence for that redo log.

Data files

Oracle datafiles store the data from tablespaces (see below). Using the tablespace name as the root of the data file unambiguously shows the connection between tablespace and data file. There are often many datafiles associated with a single dataspace so a distinguishing key (usually 2 characters) is added to the tablespace name to complete the basename.

Personal Preferences

The examples here are intended to inform, not dictate, naming. Many variations on the themes expressed here will work. Meeting the requirements described in this appendix is more important than following the rules. The examples below show variations that meet the requirements described earlier and also meet the two important tests below:

- The directory's actual name does not mater, as long as it is both consistent with the names of other similar directories, and chosen carefully to not misrepresent the contents of the directory.
- I/O balanced files can be stored at any level below the mount point, as long as it's the same level on every mount point.

Examples:

```
/u01/ORACLE/sab/system01.dbf
/disk4/oradata/pdnt/system01.dbf
/db016/ora/mail/system01.dbf
/neptune/data/disk31/oracle/bnr1/system01.dbf
/u08/app/oracle/data/pfin/system01.dbf
```

IV. Oracle Tablespaces

The Oracle data files described above are the physical storage elements for the components of an Oracle database. A database is divided into several *tablespaces*. Each table (data segment), index segment, rollback segment, and temporary segment in the database is contained in a tablespace. A tablespace, in turn, is made up of one or more data files. This section describes how to assign database objects to tablespaces.

Segment Partitioning

Factors that affect decisions about separating Oracle segments into different tablespaces include:

- *Fragmentation character.* Dropping segments can cause free space fragmentation. Separating segments with different lifespans into different tablespaces can prevent the problems associated with tablespace free space fragmentation.

- *I/O distribution.* Assigning a segment to a tablespace also assigns that segment to the data files in that tablespace. Separating segments with competing I/O requirements into different tablespaces can establish well-balanced I/O loads across hardware components.
- *Administrative needs.* The tablespace is the administrative unit to specify parts of the database to backup, or to identify the maximum size for the database of a particular user. Separating segments with different administrative characteristics into different tablespaces can give the administrator the appropriate level of control over collections of segments.

Requirement 11. Tablespace contents must be separated to (a) minimize tablespace free space fragmentation, (b) minimize I/O request contention, and (c) maximize administrative flexibility.

The next rule describes how this requirement can be met.

OFA 7

Separate groups of segments with different lifespans, I/O request demands, and backup frequencies among different tablespaces. For each Oracle database, create the following special tablespaces in addition to those needed for application segments:

SYSTEM	data dictionary segments only
TEMP	temporary segments only
RBS	rollback segments only
TOOLS	general-purpose tools only
USERS	miscellaneous user segments

This standard has proven itself at many sites. For example, because dictionary segments are never dropped, and because no other segments that can be dropped are allowed in the **SYSTEM** tablespace, this scheme guarantees that the **SYSTEM** tablespace will never require a rebuild due to tablespace free space fragmentation. Because no rollback segment is stored in any tablespace holding applications data, the administrator is never blocked from taking an applications tablespace off-line for maintenance. Because segments are partitioned physically by type, the administrator can record and predict data volume growth rates without complicated tools.

Tablespace Names

The OFA standard of embedding the name of a tablespace in the basename of its associated data files (OFA 6) means that UNIX file name length restrictions (14 characters for portable UNIX) also restrict tablespace name lengths. The datafiles include a six character suffix, so 8 characters are left for the tablespace name.

OFA 8

Name tablespaces descriptively with 8 or fewer characters.

Descriptive names allows the administrator to easily associate the name of a data file with the tablespace that uses it. For example, the names GLD and GLX might be used for the tablespaces storing General Ledger data and indexes, respectively. Figure 3 illustrates this relationship between tablespaces and data files.

V. Using Raw Partitions (devices) for Database Files

Character special (raw) files can be used to hold Oracle database files. Raw files bypass the file system, going directly to the disk driver. Until this section, we have assumed that all of the database files are part of the file system. Tip 40 describes some of the trade-offs and tuning issues associated with the use of raw partitions.

Some sites have little or no choice about the use of raw devices. The Oracle Parallel Server option requires that all database files be on raw partitions. Some platforms offer asynchronous I/O to boost write performance, but only for raw devices (Tip 31).

Other sites may chose to use raw devices for only some of their database files. Small tables that will normally live in the buffer cache, or tables that are infrequently accessed or used mainly for read access can be left in the file system. In this way, the administrative price for raw devices is only paid for tables that get the most benefit. Technically, the decision to use raw partitions can be made for each database file. For those platforms that offer asynchronous I/O to raw devices only, the Oracle Server will optimize writes, using asynchronous I/O for raw datafiles, and synchronous I/O to the file system datafiles. However, in an OFA compliant database following *Requirement 11* means making a tablespace either all raw or all file system based.

Raw redo logs

Because raw devices are most beneficial for files that do sequential writes, redo logs are ideal candidates for raw devices. In addition, on-line redo log files are not usually included in normal operating system backup procedures, so one of the primary administrative challenges for raw devices is removed

There are two additional administrative issues associated with the use of raw partitions:

- *Backup:* the only standard UNIX utility to backup a raw device is *"dd"*.
- *Fragmentation:* By definition, the file size of a raw partition equals the partition size. Once created, disk partitions are very hard to change and the size choices available may be very limited. Putting two facts together, there are

seldom enough raw partitions available when needed, and those that are available are usually of the wrong size.

Both of these issues must be addressed at any site that plans to use raw devices. Fortunately, substantial help is (or soon will be) available for many UNIX platforms. Backup issues can be avoided by only putting redo logs on raw partitions, or addressed directly using the Oracle backup offering as it becomes available.

If you have decided to use raw devices, then you will need to meet the following requirement:

Requirement 12. It must be possible to tune disk I/O load across all drives, including drives storing Oracle data in raw devices.

OFA teaches that whatever initial configuration is chosen, over time, that configuration will change significantly. OFA suggests rules and requirements that ensure adaptability for data files in the file system. The advice in the section below makes using raw devices as flexible as possible. This is also the way that the Oracle customer benchmarking team sets up a benchmark database on raw devices.

OFA 9

Chose a small set of standard sizes for all raw devices that may be used to store Oracle database files.

In general, standardizing on a single size is the way to go. If a single size is used, raw files can be moved from one partition to another safely. The size should be small enough so that a fairly large number can be created. but large enough to be convenient. For example, a two gigabyte drive could be divided into 10 partitions of 200 megabytes each -- a good balance between size and number. Any tablespace that uses raw devices should stripe them across several drives. This should be done with a logical volume manager.

Logical Volume Manager

A logical volume manager should be used, if available, to build up pre-striped raw database files. Tip 28 describes use of an LVM to meet *Requirement 12.*

For example, you could use the LVM to build 8 logical volumes striped across five disks (ten if mirroring is required for availability). This gives 8 pre-striped logical volumes of one gigabyte each ready to be assigned to datafiles. In addition, you could create 10 unstriped logical volumes for the **SYSTEM, TOOLS**, and other small tablespaces for an OPS based application.

Some logical volume managers, such as HP's, allow mirrored logical volumes to be established in increments of 4 megabytes after the striped *volume groups* are built. This allows far more precision in matching file sizes to database needs. However, sticking to a few sizes each an integral multiple of the others is still required to allow files to be moved conveniently.

Integrate raw devices with the file system using symbolic links

Figure 5 shows how symbolic links with a meaningful name can be used in an OFA compliant data file structure to point to each raw partition. Because the size of each data file is fixed, the size is embedded in the name, in this case using a two digit number just before the "." where the LSB represents 100 megabytes.

Additional raw partitions or LVM partitions will be needed from time to time, often unpredictably. Figure 4 shows that every partition available for use is included in the path */pm/q/***spare_pool** using symbolic links to connect the partition to the raw device name.

In order to perform backups or other administration that requires the knowledge of what data is on what disk or partition, create a text file with symbolic links for every datafile in every tablespace. There should be mappings to both the logical volumes and also to the physical partitions for recovery and reconfiguration. This file can be processed using the usual UNIX tools to build the scripts needed for many administrative activities.

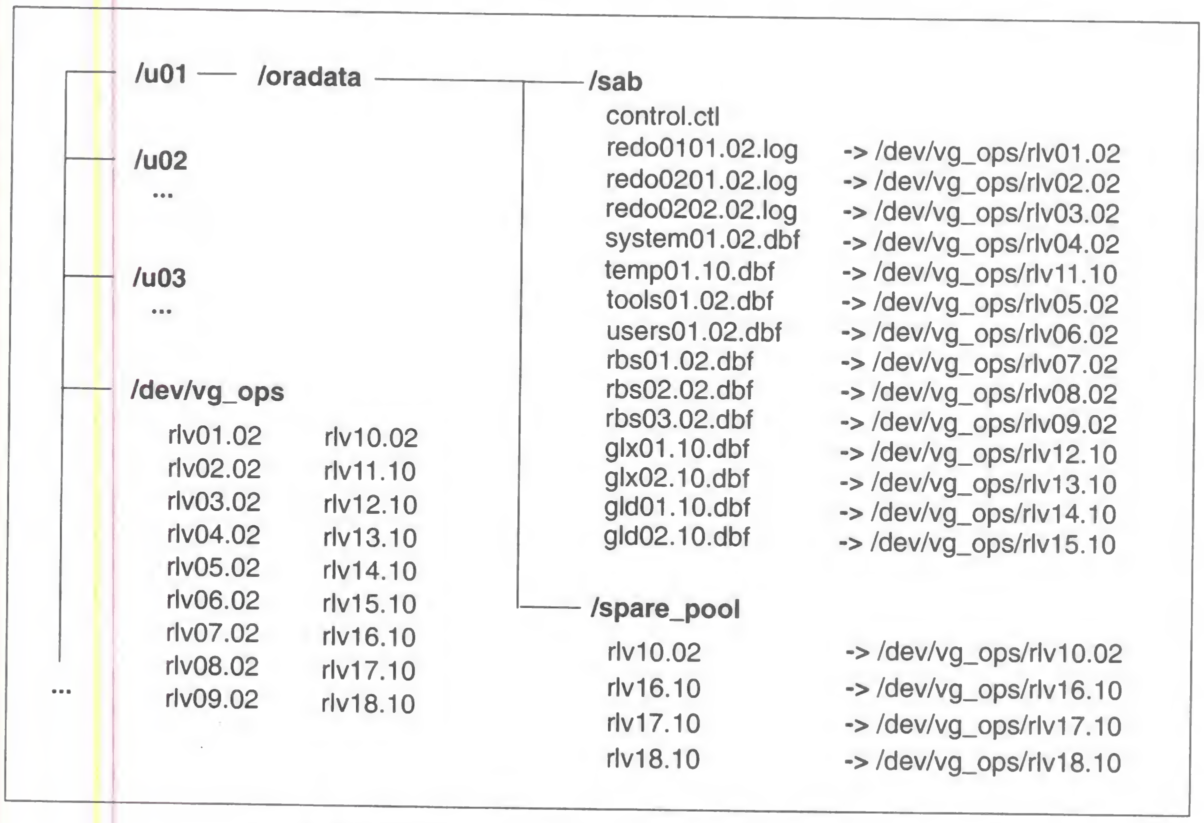

```
├── /u01 ── /oradata ──┬── /sab
│                      │     control.ctl
│                      │     redo0101.02.log    -> /dev/vg_ops/rlv01.02
├── /u02               │     redo0201.02.log    -> /dev/vg_ops/rlv02.02
│     ...              │     redo0202.02.log    -> /dev/vg_ops/rlv03.02
│                      │     system01.02.dbf    -> /dev/vg_ops/rlv04.02
├── /u03               │     temp01.10.dbf      -> /dev/vg_ops/rlv11.10
│     ...              │     tools01.02.dbf     -> /dev/vg_ops/rlv05.02
│                      │     users01.02.dbf     -> /dev/vg_ops/rlv06.02
├── /dev/vg_ops        │     rbs01.02.dbf       -> /dev/vg_ops/rlv07.02
│     rlv01.02  rlv10.02     rbs02.02.dbf       -> /dev/vg_ops/rlv08.02
│     rlv02.02  rlv11.10     rbs03.02.dbf       -> /dev/vg_ops/rlv09.02
│     rlv03.02  rlv12.10     glx01.10.dbf       -> /dev/vg_ops/rlv12.10
│     rlv04.02  rlv13.10     glx02.10.dbf       -> /dev/vg_ops/rlv13.10
│     rlv05.02  rlv14.10     gld01.10.dbf       -> /dev/vg_ops/rlv14.10
│     rlv06.02  rlv15.10     gld02.10.dbf       -> /dev/vg_ops/rlv15.10
│     rlv07.02  rlv16.10
│     rlv08.02  rlv17.10 └── /spare_pool
...   rlv09.02  rlv18.10       rlv10.02         -> /dev/vg_ops/rlv10.02
                               rlv16.10         -> /dev/vg_ops/rlv16.10
                               rlv17.10         -> /dev/vg_ops/rlv17.10
                               rlv18.10         -> /dev/vg_ops/rlv18.10
```

Figure 5: RAW devices in an OFA-Compliant Database file structure.

VI. Parallel Server Administrative Files

The Oracle Parallel Server (OPS) allows two or more independent computers, each with its own processor, memory and independent copy of the operating system to work together on the same database. Oracle distinguishes between an *instance*: one self-contained copy of the processes and shared memory that make up the database sever, and a *database*: a set of tables, indexes, control files, redo logs, etc. spread across a set of disk drives. Normally, there is a one-to-one correspondence between an instance and a database. OPS allows multiple instances to work together on a common database. For example, assume that the **sab** database is simultaneously held open by two OPS instances on two UINX nodes, **sab1** on **node1** and **sab2** on **node2**. Because the database files are shared, including the SYSTEM tablespace, a report on a data dictionary table will provide the same answer from either node. However, a report produced from a dynamic table (or view, such as any **v$parameter**) will report on the information in the local instance only. This motivates the following requirement:

> *Requirement 13.* (a) Administrative data must be stored in a central place accessible to all database administrators; and (b) instance specific administrative data is associated with a given instance by a file name.

Meeting *Requirement 13* can take advantage of the following features of the OFA standard: the directories **arch**, **create**, and **exp** are database administrative directories; **bdump**, **cdump**, **pfile**, and **udump** are instance administrative directories, and **adhoc** and **logbook** are mixtures of both.

One way to structure the administrative directory for the sab database is shown in Figure 6. In this example each administrative subtree that holds instance specific information adds a directory layer to put the information for each instance into its own directory path. NFS mounting some of the files to give a common link from all instances, appropriate use of symbolic links for admin/d, or use of a common working directory by setting it to **~oracle/admin/sab** are all options for fulfilling *Requirement 13*.

OFA 10

If you are using Oracle Parallel Server, select exactly one node to act as Oracle *administrative home* for the cluster to house the administrative subtree defined in OFA 5. Create a directory named for each instance accessing the database within the **bdump**, **cdump**, **logbook**, **pfile**, and **udump** directories of ~**/admin**/d Ensure that the **admin** directory for the *administrative home* is mounted as the **admin** directory for every instance.

```
├── /n01 – /app – /oracle – /admin — /sab/
│                                     adhoc/
├── /u02                              arch/
│    ...                                  redo0001.arc
│                                     adump/
├── /u03                                  sab1/
│    ...                                  sab2/
│                                     bdump/
│                                         sab1/
│                                         sab2/
│                                     cdump/
│                                         sab1/
│                                         sab2/
│                                     create/
│                                         1-rdbms.sql
│                                     exp/
│                                         940920full.dmp
│                                         import/
│                                         export/
│                                     logbook/
│                                         sab1/
│                                             params.1st
...                                       sab2/
                                              params.1st
                                          users.1st
                                      pfile/
                                          sab1/
                                              initsab1.ora
                                          sab2/
                                              initsab2.ora
                                      udump/
                                          sab1/
                                          sab2/
```

Figure 6: OPS **admin** directory example

Appendix 2: The utlbstat/utlestat.sql Reference

The `utlbstat/utlestat` SQL scripts (`BSTAT/ESTAT` in Oracle6) are useful tools for monitoring Oracle RDBMS performance.The scripts can be used to capture a snapshot of database performance statistics.The scripts are located in the `$ORACLE_HOME/rdbms/admin` directory and should be executed after the database has been running for a while. (The `init.ora` parameter `TIMED_STATISTICS` must be set for these scripts to work.)

To initiate the collection of statistics, navigate to the `$ORACLE_HOME/rdbms/admin` directory and start `SQLDBA`; at the `SQLDBA` prompt, start `utlbstat` by typing `@utlbstat`. `Utlbstat` creates a set of tables and views in the SYS account; these tables contain a snapshot of the state of the system at the beginning of the performance test. The database tables created are listed in Table 1.

TABLE 1. Tables created by script

View/Table Name	Description
`stats$begin_stats`	General stats from `v$sysstat`
`stats$file_view`	View of File I/O statistics
`stats$begin_file`	Table of File I/O `statistics from stats$file_view`
`stats$begin_latch`	Latch statistics from `v$latch`
`stats$begin_roll`	Rollback segment statistics from `v$rollstat`
`stats$begin_kqrst`	Dictionary Cache statistics from `x$kqrst`
`stats$dates`	Table with beginning date

After you run the application that you wish to monitor, to end the collection of statistics, run the `utlestat` script. Like `utlbstat`, `utlestat` creates a set of tables and views in the SYS account. These tables contain a snapshot of the state of the system at the end of the performance test. The tables that reflect the 'end' state of the system are listed in Table 2.

TABLE 2. End tables created by ESTAT

View/Table Name	Description
`stats$end_stats`	General stats from `v$sysstat`
`stats$end_file`	Table of File I/O statistics from `stats$file_view`
`stats$end_latch`	Latch statistics from `v$latch`
`stats$end_roll`	Rollback segment statistics from `v$rollstat`
`stats$end_kqrst`	Dictionary Cache statistics from `x$kqrst`

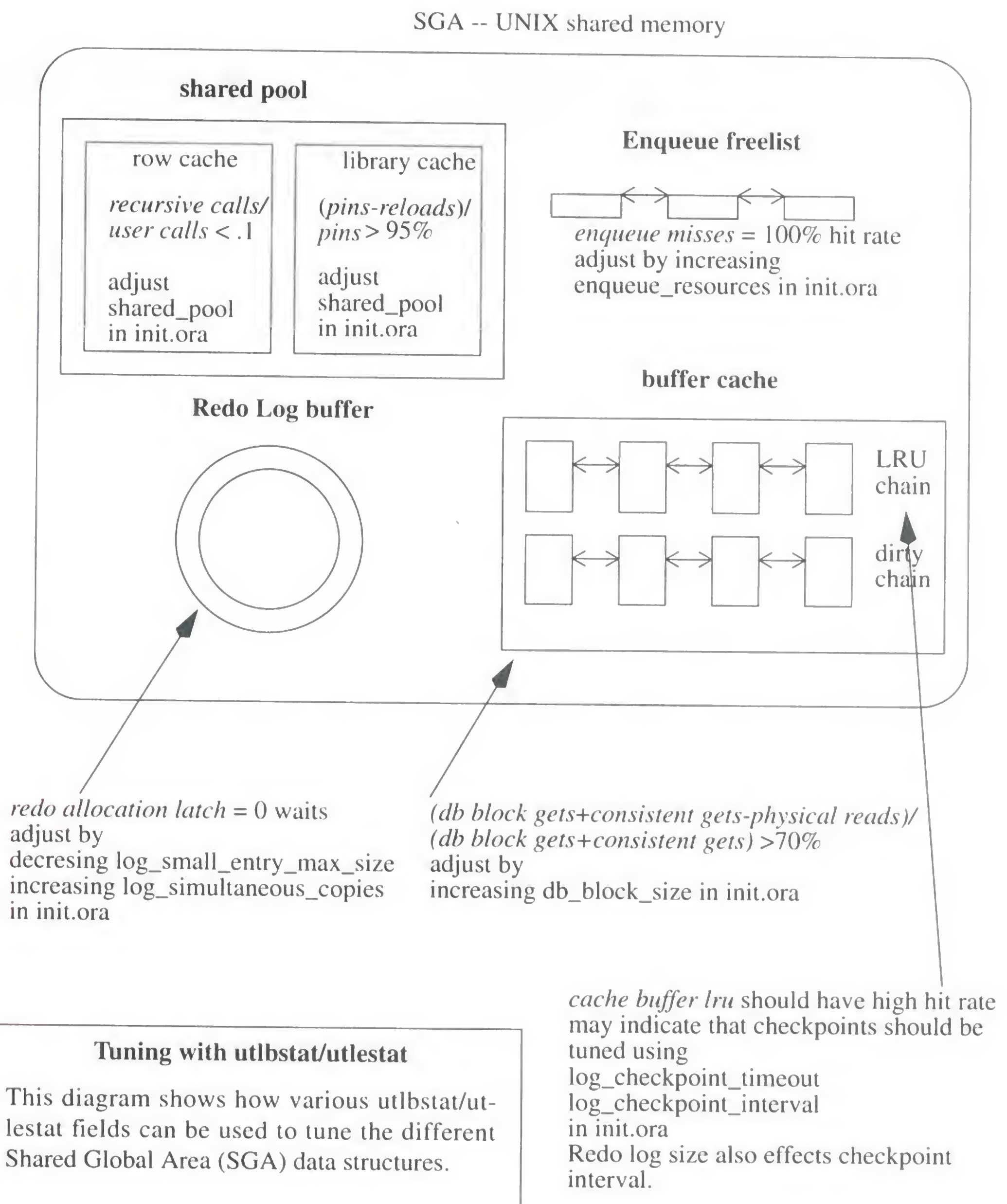
In addition to the 'end' state tables, ESTAT creates a set of tables in the SYS account which outline the differences between the 'beginning' state and 'end' state statistics. The database tables created are listed in Table 3.

TABLE 3. Statistics tables created by ESTAT

View/Table Name	Description
`stats$stats`	General system statistics
`stats$file`	File I/O statistics
`stats$latches`	Latching statistics
`stats$roll`	Rollback Segment statistics
`stats$kqrst`	Dictionary Cache statistics
`stats$dates`	Table with ending date

Finally, `utlestat` produces a report in the current directory with performance statistics from the tables listed above. The report includes 7 sections:

1. System-wide statistics.
2. File I/O statistics.
3. Latch statistics.
4. Rollback segment statistics.
5. Dictionary cache statistics.
6. INIT.ORA parameter listing.
7. Date of `utlbstat` and `utlestat` execution.


SGA -- UNIX shared memory
shared pool
row cache
recursive calls/ user calls < .1
adjust shared_pool in init.ora
library cache
(pins-reloads)/ pins > 95%
adjust shared_pool in init.ora
Enqueue freelist
enqueue misses = 100% hit rate
adjust by increasing enqueue_resources in init.ora
Redo Log buffer
buffer cache
LRU chain
dirty chain
redo allocation latch = 0 waits
adjust by
decresing log_small_entry_max_size
increasing log_simultaneous_copies
in init.ora
(db block gets+consistent gets-physical reads)/ (db block gets+consistent gets) >70%
adjust by
increasing db_block_size in init.ora
cache buffer lru should have high hit rate
may indicate that checkpoints should be
tuned using
log_checkpoint_timeout
log_checkpoint_interval
in init.ora
Redo log size also effects checkpoint
interval.

Tuning with utlbstat/utlestat

This diagram shows how various utlbstat/utlestat fields can be used to tune the different Shared Global Area (SGA) data structures.

Appendix 3: Changing Segment Tablespaces

As was discussed in the Tuning Disk I/O section, balancing your system's I/O loads may require the transfer of segments from one tablespace to another. To correct a misplacement of segments into tablespaces you can use Oracle's *export* and *import* utilities. The operation does not require you to rebuild your databases.

To move a group of segments owned by user *u* from tablespace *old* to tablespace *new*:

1. Record grants made on *u*'s segments to other users.
2. Revoke DBA privilege from *u*.
3. Revoke Form I (database) resource from *u*.
4. Revoke Form II resource on *old* from *u*.
5. Export the segments with a user-mode export.
6. Drop the segments.
7. Alter user *u* to have default tablespace *new*.
8. Grant Form II resource on *new* to *u*.
9. Create and populate the segments with a user-mode import.
10. If appropriate, grant DBA privilege to *u*.
11. Grant the appropriate privileges on *u*'s segments to other users.

Appendix 4: Separating Data and Index Segments

As was discussed in the Tuning Disk I/O section, separating indexes from their associated data segments can increase performance on large multi-drive systems. Remember that the number of database files that can be used is limited by the operating system-specific **maxdatafiles**.

To move the indexes owned by user *u* from tablespace *old* into tablespace *new*:

1. Calculate the size needed to hold the indexes and create new.
2. Grant resource on *new* to *u*.
3. Export the segments owned by *u* by specifying **full=n**, **rows=n**, and **indexes=y**.
4. Drop the indexes owned by *u*.
5. Import the segments owned by *u* by specifying **full=n**, **rows=n**, **indexes=y**, and **indexfile=***file*.
6. Edit file; change "*old*" to "*new*".
7. Execute file from SQL*Plus.

Bibliography

No one can know everything about the Oracle database, UNIX and how to tune them. This bibliography identifies a number of sources for tuning information. It is recommended that the Oracle manuals be available, at least as a reference (you might consider using the CD-ROM version). In addition, everyone responsible for tuning the Oracle Server should own several of the books from outside Oracle.

1. *Oracle7 Server Administrator's Guide*; Oracle Part A20322-2, Release 7.2.

   The DBA's most important document. Describes how to build and maintain an Oracle database. Most of the tuning advice has been moved to *Oracle 7 Server Tuning*, described below.

2. *Oracle7 Server Application Developer's Guide*; Oracle Part A20323-2, Release 7.2.

   This manual describes how to design an application. It provides an introduction to components of the server that most impact performance. Specific tuning advice is in the *Oracle 7 Server Tuning* manual, below.

3. *Oracle7 Server Tuning*, Release 7.2; Oracle Part #A25421-1.

   This manual describes SQL Processing, including Discrete Transactions, the query Optimizer, and parallel query. Each of these subjects is described in detail, along with how to use these features for best results.

4. *Oracle7 Server Reference*, Release 7.2; Oracle Part #A20327-2.

   Describes initialization parameters, data dictionary - including the V$tables, and SQL scripts needed to administer and monitor the database.

5. *Oracle7 Server Concepts Manual*, Release 7.2; Oracle Part A20321-2.

   Tuners should be generally familiar with the database concepts explained in this book.

6. *Oracle Server Manager User's Guide*; Oracle Part A30887-1.

   This manual describes use of the Server Manager. Several of the screens shown in this document are taken from server manager displays.

7. *Oracle7 Server Utilities User's Guide*; Oracle Part #3602-70-1292.

   Chapters 10 through 13 describe SQL*DBA.

8. *Oracle7 Parallel Server*, Release 7.2; Oracle Part A19487-2.

   Introduction to the parallel server option. Chapter 3, Database Design, discusses a number of parallel server tuning issues to consider when designing an OPS database.

9. *Oracle 7 Parallel Server Administrator's Guide;* Oracle Part A12852-1.

Tuning information for the Parallel Server Option is in Appendix D.

10. *Oracle Product Documentation Library;* Oracle Part A14538-10. (May, 1995)

    A CD-ROM with all of the manuals above (and many more) in Oracle Book format. An Oracle Book reader for MS-Windows 3.1 is included.

11. *Oracle for UNIX Installation and Configuration Guide* (UNIX platform-specific).

    These guides often have extensive capacity planning information including platform specific rules-of-thumb. In addition, additional tuning information for Step 6: Tuning CPU Usage, and Step 7: Tuning Resource Contention is often available.

12. Oracle Consulting: Core Technologies Team, *An Optimal Flexible Architecture for a Growing Oracle Database*, Installing Oracle for Optimal System Performance; Oracle Part # A19308.

    Additional OFA material beyond what is in Appendix 1.

13. , Oracle Consulting: Core Technologies Team, *Administrative and Performance Suite (APS)*, Curator: Craig A. Shallahamer; email: cshallah@us.oracle.com.

    A set of useful SQL scripts for Oracle DBA's.

14. *Oracle7 for AIX, Performance Tuning Tips;* Oracle Part A32148-1.

15. *Oracle7 for Sun, Performance Tuning Tips*; Oracle Part No. A25584-1.

16. *Oacle7 for UnixWare: Performance Tuning Tips*; Oracle Part No. A18831-1.

17. Peter Corrigan and Mark Gurry; *Oracle Performance Tuning*; O'Reilly & Associates, Inc.; Sebastopol, CA; 1993

    596 Pages of Oracle Performance Tuning advice. A fine collection of Oracle tuning information. Caution: the exact syntax of some of the examples may need some adjusting before they will work on your system or revision of the Oracle Server. (At least this was our experience.)

18. Michael J. Corey, Michael Appey, Daniel J. Dechinchio, Jr.; *Tuning Oracle*; Oracle Press division of Osborne McGraw-Hill; Berkeley, CA, 1995

    Edited by Oracle, this tuning book includes information about utlbstat/utlestat and other features not documented in current Oracle documents. Integrates descriptions about how the Oracle Server works with tuning advice. SQL statement tuning and parallel system tuning not covered in as much depth as the *Oracle7 Server Tuning* book.

19. Steven M. Bobrowski; *Mastering Oracle7 & Client-Server Computing*; Sybex, ISBN 0-7821-1353-2.

20. Kevin Loney; *Oracle DBA Handbook*; Oracle Press division of Osborne McGraw-Hill; Berkeley, CA, 1994

    Missing some of the 7.2 information, this book is still a very good reference volume for a DBA. Chapter 8, on tuning offers tuning suggestions often overlooked, like use of parallel SQL*Loader. His chapter 3, Logical Database Layouts uses an early version of OFA as a base.

21. Mike Loukides; *System Performance Tuning*; O'Reilly & Associates; Sebastopol, CA; 1991.

The "fish book", this document describes the tools available on most UNIX systems. The second non-Oracle document to own.

The next six books are popular books covering UNIX system administration, UNIX internals, and computer architecture. New additions appear on a regular basis.

22. Evi Nemeth, Garth Snyder, and Scott Seebass; *UNIX System Administration Handbook*; Prentice-Hall; Englewood Cliffs, NJ; 1989.

23. Aeleen Frisch; *Essential System Administration*; O'Reilly & Associates; Sebastopol, CA; 1991.

24. Maurice J. Bach; *The Design of the UNIX Operating System*; Prentice-Hall; Englewood Cliffs, NJ; 1986.

25. Bernard M. Goodheart, James Cox; *The Magic Garden Explained*; Prentice Hall.

26. Jerry Peek, Mike Loukides, Tim O'Reilly et al.; *Unix Power Tools*; O'Reilly & Associates.

27. David A. Patterson and John L. Hennessy; *Computer Architecture A Quantitative Approach*; Morgan Kaufmann Publishers; San Mateo, CA; 1990.

    An excellent book on computer architecture. The examples, methods and historical insights are unrivalled. The writing is clear, clean, well organized and persuasive. Patterson and Hennessy have also published a new text focusing on the interaction of architecture and operating systems.

28. *VAX VMS Tuning for Oracle*; Oracle Part #52569.0992.

Reader's Comment Form

ORACLE for UNIX Performance Tuning Tips

Part No. A22535-2

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication.Your input is an important part of the information used for revision. You can also send your comments electronically to `mhjohnso@oracle.com`.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any suggestions for improvement, please indicate the topic, chapter, and page number below:

Please send your comments to:

Open Systems Division

attn: Mark H. Johnson
500 Oracle Parkway,
Box 659103
Redwood Shores, CA 94065, USA

If you would like a reply, please give your name, address, and telephone number below:

Thank you for helping us improve our documentation.

A22535-2

ORACLE®